Novel View Prediction Error as a Quality Metric for Image-Based Modeling and Rendering

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs (Dr.-Ing.) von

Michael Wächter, M.Sc.,

geboren in Schwerin.

Referenten: Prof. Dr.-Ing. Michael Goesele
Technische Universität Darmstadt
Prof. Daniel Scharstein, PhD
Middlebury College, Middlebury, VT, USA

Tag der Einreichung: 06.07.2017
Tag der Disputation: 18.08.2017

Darmstädter Dissertation, 2017
D17

Ich widme diese Arbeit meiner Familie – Heimfried, Irina, Johanna, Victoria, Nike, Alexa und ganz besonders natürlich Chrisi. Ich danke euch allen so sehr für eure Großherzigkeit.

Je n'ai fait celle-ci plus longue que parce que je n'ai pas eu le loisir de la faire plus courte.

— Blaise Pascal, 1657

Abstract

Image-based modeling and rendering (IBMR) is a sub-discipline of visual computing whose objective it is to capture images of a scene in the real world, construct a model of the world using the captured image data, and use this model to synthesize images of the world from previously unobserved viewpoints. This so-called novel (or virtual) view prediction has traditionally been tackled from two sides: On one side the computer vision community has pursued the construction of geometric models from sets of images only. On the other side the computer graphics community has worked on producing photo-realistic renderings from hand-modeled, virtual scenes and has further come up with algorithms that allow for the synthesis of novel views from input photos of real-world scenes either directly without any geometric models or with approximate, hand-modeled geometry models. The wealth of different IBMR systems also brought in its wake various quality evaluation systems that are more or less tailored to the properties of specific IBMR systems. In recent years, computer vision and graphics have grown together, slowly approaching the goal of novel view prediction on scenes without restrictions. However, the fragmentation of evaluation systems has still not been overcome.

This thesis makes two main complementary contributions: We first present a novel texture mapping algorithm that assigns a static texture to polygonal 3D models, given images that are registered in the same coordinate frame as the model. Our texturing algorithm takes into consideration real-world scenes' properties such as illumination and exposure changes between images, non-rigid scene parts, unreconstructed occluders such as pedestrians, and images with pixel footprints that vary by orders of magnitude. We address the size (*i.e.*, the number of images and the number of polygons in the geometry model) of real-world datasets with a novel Markov random field solver that solves the main bottleneck of our texturing framework orders of magnitude faster than related work. Conceptually, we can think of our texturing framework as closing the gap between image-based 3D reconstruction and photo-realistic rendering, thereby turning 3D reconstructions into full-fledged IBMR representations.

Second, we introduce an evaluation scheme for IBMR methods that is guided by the definition of IBMR: Novel view prediction error evaluates how well an IBMR algorithm predicts novel views by dividing all input images into training and test images, keeping the test images secret, giving the training images to the IBMR algorithm, letting it predict the test images, and comparing its predictions with the actual test images. In this thesis we verify that (if used in conjunction with suitable image comparison metrics) this scheme fulfills a range of basic, intuitive conditions. We further compare our scheme with traditional, geometric 3D reconstruction eval-

uation schemes, show in a user study how our scheme relates to human judgment of the quality of novel view predictions, and present a new, general IBMR benchmark based on our evaluation scheme.

Zusammenfassung

Image-based Modeling und Rendering (IBMR) ist eine Teildisziplin des Visual Computing, deren Ziel es ist Bilder in der Welt aufzunehmen, daraus ein virtuelles Modell der Welt zu konstruieren und dieses Modell zu benutzen, um Bilder der Welt aus der Sicht neuer Blickpunkte zu synthetisieren. Dieses Problem der sogenannten Novel (oder Virtual) View Prediction wurde traditionell von zwei Seiten angegangen: Auf der einen Seite haben Computer Vision-Forscher daran gearbeitet, die automatische Rekonstruktion von geometrischen Modellen nur basierend auf Photos zu ermöglichen. Auf der anderen Seite haben Computergraphiker daran gearbeitet, handmodellierte virtuelle Szenen photorealistisch zu rendern, und des Weiteren Algorithmen entwickelt, die das direkte Synthetisieren von Novel Views aus Eingabebildern entweder direkt ohne Geometriemodell oder unter Zuhilfenahme grober, handmodellierter Geometriemodelle ermöglichen. Die Vielfalt an unterschiedlichen IBMR-Systemen hat auch eine Vielfalt an Qualitätsevaluationssystemen, die mehr oder minder stark an die Eigenheiten der jeweiligen IBMR-Systeme angepasst sind, mit sich gebracht. In den letzten Jahren sind Computer Vision und Computergraphik zusammengewachsen und nähern sich langsam dem Ziel an, realistische Novel Views in Szenen ohne Restriktionen synthetisieren zu können. Die Fragmentierung der Evaluationssysteme ist bisher jedoch nicht überwunden worden.

Diese Arbeit leistet zwei komplementäre Hauptbeiträge: Zum Ersten präsentieren wir einen neuartigen Texture-Mapping-Algorithmus, der, gegeben ein Geometriemodell und gegen das Modell registrierte Bilder, das Modell mit den Bildern texturiert. Unser Algorithmus betrachtet dabei die Eigenschaften von Echtwelt-Szenen wie unterschiedliche Beleuchtung und Belichtung zwischen Bildern, bewegliche Szenenteile, nicht rekonstruierte Okkluder wie zum Beispiel Fußgänger und drastische Unterschiede zwischen den Pixel-Footprints verschiedener Bilder. Die Größe von Echtweltdatensätzen gehen wir mit einem neuartigen Markov-Random-Field-Solver an, der das Hauptnadelöhr unseres Texturierungsframeworks um Größenordnungen schneller löst als verwandte Arbeiten. Konzeptionell kann unser Texturierungsframework so betrachtet werden, dass es die Lücke zwischen bildbasierter 3D-Rekonstruktion und photorealistischem Rendering schließt und damit 3D-Rekonstruktionen zu vollwertigen IBMR-Repräsentationen macht.

Zum Zweiten führen wir eine Evaluationsmethode für IBMR-Repräsentationen ein, die sich die Definition von IBMR zunutze macht: Der Novel-View-Prediction-Fehler bewertet, wie gut ein IBMR-Algorithmus darin ist, Novel Views zu synthetisieren, indem man alle Eingabebilder in Trainings- und Testbilder aufteilt, die Testbilder geheim hält, die Trainingsbilder dem IBMR-Algorithmus zur Verfügung stellt, ihn die Novel Views vorhersagen lässt und seine Vorhersagen mit den geheimen

Testbildern vergleicht. In dieser Arbeit verifizieren wir, dass diese Vorgehensweise (in Kombination mit geeigneten Bildvergleichsmetriken) gewisse grundlegende Eigenschaften erfüllt. Des Weiteren vergleichen wir unsere Evaluationsmethode mit traditionellen, geometrischen 3D-Rekonstruktionsevaluationsmethoden, zeigen in einer Benutzerstudie wie sich unsere Methode zu menschlichen Einschätzungen der Qualität von Novel-View-Predictions verhält und präsentieren einen neuen, generalisierten IBMR-Benchmark, der auf unserer Methode basiert.

Acknowledgments

Foremost, I want to express my deepest gratitude to my family: Heimfried, Irina, Johanna, Victoria, Nike, Alexa, and my brother Chris. I cannot thank you enough for being there.

Also, a very big thank you to my advisor Michael Goesele. As “Doktorvater” (“doctoral father”) he always gave us “kids” enough freedom to pursue what we were interested in and was always there when we needed help. He helped me mature academically, supported my internship and post-doc plans, and countless more things. Thank you for being my mentor, Michael.

I am very grateful to Daniel Scharstein, not only for kindly agreeing to review this thesis, but also for providing us with the Middlebury MView benchmark submissions. Thanks as well to Rick Szeliski for valuable discussions about the virtual rephotography paper and for inspiring discussions in Zürich/Seattle/Amsterdam, to my co-authors such as Johannes Kopf for being very helpful in improving our papers, to Douglas Cunningham for giving us advice on the rephoto evaluation, and to my Microsoft Research mentor Eyal Ofek whose ideas and speed of thinking were truly inspiring.

Many thanks to my colleagues, to Uschi with whom I had lots of funny moments in the office and on the way home, to Jens, Simon, Fabian, André, Dom, Martin, Daniel 1, Samir, Nico, Xiang, and our immensely helpful minions Flo, Benjamin, Daniel 4, and Stepan. Some of you also belong into the friends list below. And I am sure I forgot many people. Please feel included in this list.

Thank you to my friends Jonas, Mate, Dirk, Daniel 2, Tobi, and of course Nils, for bringing so much happiness into my life.

And finally a big, big thank you to Chris and Ishy for proof-reading all of this!

Contents

1. Introduction

Image-based modeling and rendering (IBMR) is concerned with capturing and realistically reproducing the visual appearance of the real world. This is demonstrated in Figure 1.1: The first row shows photos of a scene in Darmstadt and the second row shows top-down renderings of a 3D reconstruction of that scene,[1] which was reconstructed from the three images above and 75 other images. The images in the second row also indicate all input images' camera positions (gray pyramids) and the viewing directions of the images in the first row (red lines). In the third row, we see a novel viewing position that was not among the input images and a prediction of what the world might look like from that position. The goal of image-based modeling and rendering is exactly this: Synthesizing novel views of a three-dimensional scene given some images from known viewpoints.

The Plenoptic Function: One way to formalize this is the so-called plenoptic function introduced by Adelson and Bergen [1991]:

$$P(\theta, \phi, \lambda, V_x, V_y, V_z, t) \tag{1.1}$$

This function describes the light intensity at wavelength λ and time t entering an eye or camera at point (V_x, V_y, V_z) from the direction (θ, ϕ). The plenoptic function is not, however, a specific function or algorithm, but should be thought of as an abstract goal definition for image-based modeling and rendering: We want to design algorithms that capture the world and then, when queried about a position in the world, a direction, a wavelength, and a point in time, return the correct light intensity.

This continuous 7D function is of course impossible to capture and store for every point, direction, wavelength, and time. Therefore, one typically makes simplifying assumptions such as eliminating time (*i.e.*, assuming a static scene and treating everything time-varying as outliers[2]) and wavelength, which yields the five-dimensional plenoptic function [Kang et al. 2006, Section 4.1]

$$P_5(\theta, \phi, V_x, V_y, V_z). \tag{1.2}$$

In the remaining function domain – viewing position and angle – we capture a set of data points and construct models that allow us to predict the rest of the function from the captured points, or as McMillan and Bishop [1995] put it:

[1] Keep in mind that this virtual scene representation is just one of many possible representations. In Section 2.3 we will also explain some fundamentally different ones.

[2] Of course, not every system makes these simplifications. Video-based rendering, for example, does look at the time as a variable of the plenoptic function.

Figure 1.1.: *1st row:* Three (out of a total of 78) images of a statue at Darmstadt's Mathildenhöhe. *2nd row:* A 3D reconstruction of the scene. Gray pyramids represent input image positions and the red lines represent the viewing directions of the images in the first row. *3rd row:* A new camera position from which we have not previously captured an image and a synthesized image for that camera position.

> "Given a set of discrete samples [...] from the plenoptic function, the goal of image-based rendering is to generate a continuous representation of that function. This problem statement provides for many avenues of exploration, such as how to optimally select sample points and how to best reconstruct a continuous function from these samples."

Samples from the plenoptic function are captured by taking photos (first row of Figure 1.1). Each pixel in a captured image is an integral over all the infinitely many rays that come from light sources in the world, somehow interact with objects in the world, and then find their way through the camera lens or pinhole and into the pixel.[3] The task of an IBMR algorithm is to take many samples, *i.e.*, images,

[3]Sometimes it helps to think of it in the opposite way: Pixels correspond to rays from the pixel

and combine them to generate novel images (bottom right of Figure 1.1) that are hypotheses for how the world looks from novel viewpoints that we have not observed so far (bottom left of Figure 1.1). In the literature, this process is called *novel view prediction* [Szeliski 1999] or *view synthesis problem* [Scharstein 1999, Fitzgibbon et al. 2005] and novel views are sometimes called *virtual cameras* or *virtual views*.

Even though the plenoptic function's behavior in between known samples is not arbitrary,[4] McMillan and Bishop's problem formulation above is very general and may, depending on the scene, involve all phenomena of real-world light transport including scattering, (partial) absorption, reflective surfaces, and many more. Therefore, this problem is far from being solved, even 22 years after its formulation.

Convergence of Graphics and Vision: In the past, the research communities of computer vision and computer graphics have tackled this topic from opposite directions: Computer graphics generates images from models and researchers have come up with more and more elaborate models and techniques that increase the realism of rendered images to the point where it is hard to tell real and rendered images apart at first glance.[5] Computer vision goes the opposite way; it takes images of the real world and tries then to infer models from them. Lengyel [1998] remarked how both fields slowly started "meeting in the middle, and the center—the prize—is to create stunning images". Computer graphics and vision have cross-fertilized each other to a great extent: Computer graphics has, for example, given computer vision fast spatial data structures, fast graphics hardware, geometry processing algorithms, computer-generated images for the training of vision algorithms (*e.g.*, Shotton et al. [2011]), and countless more. Conversely, computer vision's image-based modeling helps to automate the creation of realistic geometry. Back in 2000, Shum and Kang were still very skeptical about the abilities of computer vision regarding geometry reconstruction:

> "For real environments, these models can be generated [... by] applying computer vision techniques to captured images. Unfortunately, vision techniques are not robust enough to recover accurate 3D models."

And in 2006 Kang et al. wrote:

into the scene. This is similar to the extramission theory of vision according to which Euclid and other Greek philosophers believed that "rectilinear rays proceeding from the eye diverge infinitely [and] those things are seen upon which the visual rays fall and those things are not seen upon which the visual rays do not fall" [Gross 1999]. The theory was even disputed at its time, but nevertheless, Winer et al. [2002] found that more than 50 % of US college students think that this is how visual perception works. This theory has obvious flaws (it would, for example, enable night-time vision) for which people have come up with elaborate workarounds that would not withstand Ockham's razor. Even though we can nowadays discard this theory, thinking about the image formation process starting from the eye or camera is sometimes handy (and actually used in computer graphics techniques such as ray casting or path tracing).

[4] *E.g.*, epipolar geometry restricts the positions where a point in one image can show up in another.

[5] "Reality vs. Crysis": `http://science.trigunamedia.com/treeoflife/rvc.htm`

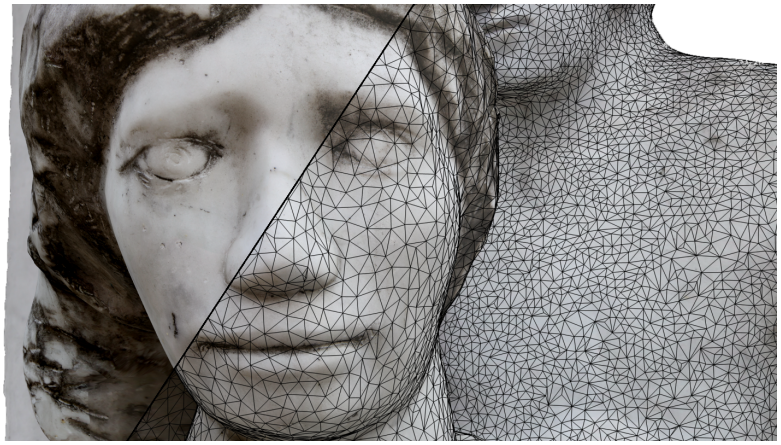


Figure 1.2.: Photo-realistic rendering of a reconstructed statue (on Darmstadt's Kopernikusplatz).

> "It is thus not surprising that despite IBR as a field having been around for some number of years, most of its representations have not been adopted for widespread commercial use. The notable exceptions are the simplest ones such as panoramas."

Meanwhile, much progress has been made and breakthroughs have been reached regarding structure from motion [Snavely et al. 2006], multi-view stereo [Goesele et al. 2007, Furukawa et al. 2010], and surface reconstruction [Kazhdan et al. 2006, Fuhrmann and Goesele 2014]. We can nowadays confidently say that computer vision is mature enough to reconstruct accurate geometry models (see examples in Figures 1.1 and 1.2) even from unstructured image collections that inexperienced users simply captured in order to share them with other internet users and not with 3D reconstruction in mind [Agarwal et al. 2009, Frahm et al. 2010, Heinly et al. 2015, Schönberger et al. 2016]. Further, we argue that – contrary to what Kang et al. wrote in 2006 – even image-based modeling and rendering techniques much more complex than panorama stitching have arrived in commercial applications such as Google Earth, commercial 3D reconstruction software[6], or computer games[7]. Even conferences in computer graphics and vision have considerable overlap nowadays, *e.g.*, with computer vision papers explicitly being invited in the call for papers of SIGGRAPH and SIGGRAPH Asia. This also lead to the wide-spread use of the phrase "visual computing" for the combination of computer graphics and vision.

Since image-based modeling deals with inverse, ill-posed problems[8], it is by its very nature inexact. Its results depend on the noise in the input images, the math-

[6] Autodesk Remake (`http://remake.autodesk.com`), PhotoScan (`www.agisoft.com`), RealityCapture (`www.capturingreality.com`), Pix4D (`www.pix4d.com`), SURE (`www.nframes.com`), Thorskan, and more.

[7] "The Vanishing of Ethan Carter" (`www.theastronauts.com/2014/03/visual-revolution-vanishing-ethan-carter`) and "Get Even" (`www.thefarm51.com/projekt/get-even`)

[8] *E.g.*, projecting 3D world points to 2D image points is well-defined but the inverse is ambiguous.

ematical models incorporated in the algorithms, *etc.* Further, as mentioned above, the plenoptic function has to be inferred from a limited set of samples. Thus, when developing IBMR algorithms or when working on a new reconstruction, the question of the algorithm or reconstruction quality naturally arises.

Most conventional approaches to IBMR evaluation focus on one single aspect of IBMR, for instance, on detecting ghosting and popping artifacts in IBR or on measuring the geometric accuracy of multi-view stereo reconstructions. One reason for this is that the subfields of IBMR – and therefore their corresponding evaluation schemes as well – partially developed independently and only grew together in the last one or two decades. This leads to a fragmentation of the field into algorithm subsets, where different subsets are not directly comparable because they require different evaluation algorithms. Renderings of texture-mapped 3D reconstructions do not need to be evaluated with ghosting or popping detectors because these artifacts do not occur here. Similarly, evaluating geometric accuracy does not make sense in IBMR algorithms that do not use geometry.

This thesis therefore proposes a unified evaluation framework for the evaluation of all IBMR algorithms that treats the problem of IBMR (excluding camera calibration) as a black box that gets images as input and produces images from novel viewpoints as output. Essentially, it evaluates how well algorithms perform in the above-mentioned task of predicting unknown points of the plenoptic function from known samples.

1.1. Contributions

In detail, the main contributions of this thesis are as follows:

- We present the first comprehensive texturing framework for large-scale, real-world 3D reconstructions. Our method addresses most challenges occurring in such reconstructions: the large number of input images, their drastically varying properties such as image scale, (out-of-focus) blur, exposure variation, and occluders such as moving plants or pedestrians. Using the proposed technique, we are able to texture datasets that are several orders of magnitude larger and far more challenging than those shown in prior work.

  [Waechter et al. 2014b], Part I Chapter 3[9]

- Further, we present a massively parallel Markov random field solver based on block-coordinate descent with tree-shaped coordinate blocks. It handles Markov random fields with tens of millions of nodes, hundreds of labels, irregular topologies, dense and sparse label sets, arbitrary smoothness costs, and

[9] In relation to the guidelines of TU Darmstadt's computer science department regarding self-quotations ("Hinweise zu Eigenzitaten in wissenschaftlichen Arbeiten des Promotionsausschusses FB Informatik der TU Darmstadt EZ-2014/10"): This chapter is an edited and extended version of the aforementioned publication and contains large verbatim quotes from the publication.

even label costs. This gives it enough flexibility to handle more MRF problems than most existing solvers. Its speed is comparable to existing solvers on medium-sized problems and orders of magnitude faster on huge problems. Most importantly, it solves a bottleneck in the above-mentioned texturing framework in a matter of seconds or minutes rather than hours, reducing the computation for texturing to acceptable runtimes.

[Thuerck et al. 2016], Part I Chapter 4[10]

- Finally, we propose a unified evaluation approach based on novel view prediction error that is able to analyze the visual quality of any IBMR method, *i.e.*, methods that can render novel views of a scene from input images. It provides a quantitative, view-based error metric in terms of image difference and completeness for the evaluation of photo-realistic renderings. A key advantage of this approach is that it does not require ground-truth geometry, which dramatically simplifies the creation of test datasets and benchmarks. It also allows us to evaluate the quality of an unknown scene during the acquisition and reconstruction process, which is useful for acquisition planning. We give a thorough evaluation of our approach on a range of IBMR methods, including standard geometry and texture pipelines and unstructured Lumigraph techniques, compare it to an existing geometry-based benchmark, and demonstrate its utility for use cases such as local reconstruction error visualization. Further, we compare our approach's reconstruction quality scores to human judgment on reconstruction quality in a user study and present a new benchmark for IBMR systems based on our evaluation technique.

[Waechter et al. 2017], Part II[9]

1.2. Thesis Overview

This thesis is structured as follows:

Chapter 2 "Background" explains various basic concepts and techniques that will be used throughout this thesis: First we explain how the process of image formation (*i.e.*, from real-world 3D scenes to projected 2D images) is modeled mathematically. We then explain how image-based modeling, *i.e.*, the reconstruction of 3D models from images, is done including feature matching, structure from motion, multi-view

[10] The work for this chapter started as Daniel Thuerck's master thesis, which was from the very beginning motivated by our texturing application with its MRF bottleneck and culminated into a paper [Thuerck et al. 2016]. The author contributed to the project by working on the algorithm's initial CPU implementation, keeping the connection to the texturing application, helping in the experimental evaluation, and writing large parts of the paper. This chapter is a completely paraphrased version of the paper, which has been distilled to what is relevant to fast texturing. We leave out aspects, such as label costs, that are only relevant to other MRF applications.

stereo, and surface reconstruction. Next, we give an overview of some basic image-based rendering methods from the point of view of the IBR classification system by Kang et al. [2006], and finally we explain Markov random fields, an important tool in computer vision, and some basic techniques to solve them.

The remainder of the thesis is divided into two parts: Since geometric 3D reconstructions from, say, multi-view stereo followed by surface reconstruction, cannot directly be rendered photo-realistically, they are not a complete IBMR representation. **Part I** of this thesis closes this gap by presenting a method that assigns a static texture to reconstructed polygonal models that is stitched from the reconstruction input. In **Part II** we come to the main part of this thesis by introducing our generalized IBMR evaluation scheme.

Part I, Chapter 3 describes our framework that textures 3D reconstructed mesh models from the reconstruction input images. We explain an algorithm from prior work that forms the basis for our algorithm, explain the modifications we made in order to make texturing work on large real-world datasets, including their many challenges such as scale differences between images or occluders such as pedestrians in images, and then we evaluate our approach in qualitative experiments that demonstrate the improvements that our algorithm's various building blocks make compared to prior work.

Part I, Chapter 4 introduces our large-scale Markov random field solver. We first derive a block-coordinate descent scheme from two algorithms from prior work, before introducing two heuristics improving the basic algorithm. Throughout this chapter we address how the individual algorithm parts can be parallelized to multi-core and even many-core systems. We then demonstrate how our solver improves the runtime of our texturing algorithm's most crucial bottleneck.

Part II, Chapters 5–8 introduce virtual rephotography, our unified evaluation methodology for image-based modeling and rendering algorithms. We postulate a set of general desiderata for evaluation methodologies, one of them being the ordering criterion, which states that representation A should receive a lower error score than B *iff* A is "better" than B. We then verify in a range of experiments that virtual rephotography does actually fulfill this trivial-to-understand yet not-so-trivial-to-verify desideratum, compare it to the geometric Middlebury benchmark for multi-view stereo reconstructions and to users' ratings of 3D reconstruction quality, demonstrate how virtual rephotography can be used to locally highlight errors in geometric 3D reconstructions, before introducing our new IBMR benchmark that is based on virtual rephotography.

Chapter 9 concludes this thesis by summarizing its contributions and highlighting avenues for future work.

2. Background

As the name suggests, image-based modeling and rendering is concerned with images captured with cameras. We will thus first explain the mathematical image formation process of perspective cameras in Section 2.1, before detailing image-based modeling in Section 2.2, and image-based rendering in Section 2.3. Finally, in Section 2.4 we give an introduction to an important and frequently used computer vision tool: Markov random fields.

Many details about these topics can also be found in the excellent textbooks of Szeliski [2010] and Hartley and Zisserman [2004]. We will stick to an abstraction level that is sufficient for understanding the later chapters of this thesis.

Notation: Before we go into details regarding image formation and image-based modeling and rendering, we give some brief remarks on mathematical notation. Throughout this thesis we try to stick to Szeliski's [2010] notation: Vectors are lower case bold ($\mathbf{v}$), matrices are upper case bold ($\mathbf{M}$), scalars are mixed case italics (a, B, γ), and homogeneous vectors have a tilde ($\tilde{\mathbf{x}}$).

Further, $\mathbf{I}_i$ is an $i \times i$ identity matrix (sometimes we may write $\mathbf{I}$ if the dimensionality is clear from the context, *e.g.*, $\mathbf{I} = \mathbf{I}_3$ for a 3D rotation matrix that does not rotate) and $\mathbf{0}_{i\times j}$ is an $i \times j$ matrix of zeros.

$\mathbf{A}^\mathsf{T}$ is the transpose of matrix $\mathbf{A}$ and $\mathbf{A}^{-\mathsf{T}}$ is the inverse of the transpose: $\mathbf{A}^{-\mathsf{T}} = (\mathbf{A}^{-1})^\mathsf{T} = (\mathbf{A}^\mathsf{T})^{-1}$.

The cross operator in matrix form $[\,]_\times$ turns a 3-vector into a matrix so we can replace a cross product with a matrix-vector multiplication:

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \cdot \mathbf{b} \text{ with}$$
$$[\mathbf{a}]_\times = \left[\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \right]_\times \stackrel{\text{def}}{=} \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix}. \tag{2.1}$$

2.1. Image Formation with Perspective Cameras

A point in the n-dimensional world is denoted as $\mathbf{x} = (x_1, \ldots, x_n)^\mathsf{T} \in \mathbb{R}^n$. Many operations on points, lines, and planes can be expressed more compactly if we instead use homogeneous coordinates, where $(x_1, \ldots, x_n)^\mathsf{T}$ is associated with a whole equivalence class of vectors $\tilde{\mathbf{x}} = \{w \cdot (x_1, \ldots, x_n, 1)^\mathsf{T} \mid w \in \mathbb{R} \setminus \{0\}\}$. We can convert

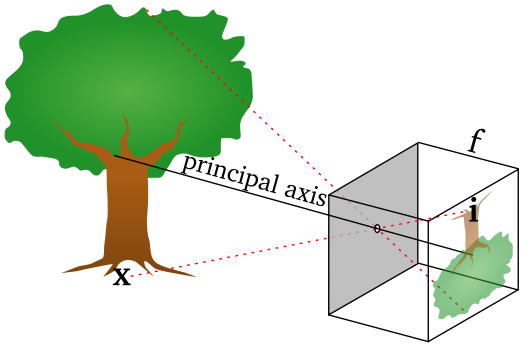


Figure 2.1.: An illustration of a pinhole camera. The image was taken and modified from Wikimedia Commons (by en:User:Pbroks13, public domain, `https://commons.wikimedia.org/wiki/File:Pinhole-camera.svg`).

back and forth between inhomogeneous and homogeneous coordinates as follows:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix} = \tilde{\mathbf{x}} \text{ and} \tag{2.2}$$

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x}_1 \\ \vdots \\ \tilde{x}_n \\ w \end{pmatrix} \mapsto \begin{pmatrix} \tilde{x}_1/w \\ \vdots \\ \tilde{x}_n/w \end{pmatrix} = \mathbf{x}. \tag{2.3}$$

It would actually be correct to write $\begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix} \in \tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}} \ni \begin{pmatrix} \tilde{x}_1 \\ \vdots \\ \tilde{x}_n \\ w \end{pmatrix}$, respectively, since $\tilde{\mathbf{x}}$ is not a single vector but an equivalence class. However, we think the equality notation is slightly clearer.

Homogeneous coordinates allow us to, for example, rotate and translate a 3D point in one single, linear operation:

$$\tilde{\mathbf{x}}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix} \cdot \tilde{\mathbf{x}}, \tag{2.4}$$

where $\mathbf{R}$ is a 3×3 rotation matrix and $\mathbf{t}$ is a 3×1 vector.

2.1.1. Pinhole Camera

A pinhole camera (see Figure 2.1) is a rectangular box with a tiny hole – the *pinhole* or *camera center* – on one end and an image plane at the other end that is a distance of f away from the pinhole. The line perpendicular to the image plane going through the pinhole is called the *principal axis*, the point where the principal axis intersects the image plane is called the *principal point*, and f is the *focal length*.

Each point $\mathbf{x}$ in the world emits rays in all directions and all rays except for the ones going through the pinhole are blocked by the plane around the pinhole. The

ray going through the pinhole falls onto point **i** on the image plane (this is shown in dashed red for the tree's top and bottom in Figure 2.1). The combination of all these points forms a horizontally and vertically flipped image of the world on the image plane. Instead of this flipped image, one often visualizes the projection process with a virtual image plane *in front* of the camera center where the image is not flipped. This can be seen in Figure 2.3.

2.1.1.1. Projection

Let us assume for a moment that our camera's pinhole is located at the world coordinate frame's origin, the principal axis "looks" along the world's z-axis and the image plane's x- and y-axis are aligned with the world's x- and y-axis. With this configuration, a homogeneous 3D point $\tilde{\mathbf{x}} = (x, y, z, 1)^\mathsf{T}$ ends up on the following point **i** in the image plane:

$$\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} fx \\ fy \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{fx}{z} \\ \frac{fy}{z} \end{pmatrix} = \mathbf{i}. \tag{2.5}$$

As we can see, a projection reduces a 3D to a 2D vector. The purpose of image-based modeling / 3D reconstruction, to which we will come back in Section 2.2, is to regain this "lost" dimension.

Principal Point: If the image plane's 2D coordinate system does not have its origin at the principal point, we need to introduce a principal point offset:

$$\begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} fx + p_x z \\ fy + p_y z \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{fx}{z} + p_x \\ \frac{fy}{z} + p_y \end{pmatrix} = \mathbf{i}. \tag{2.6}$$

Often (but not always) this principal point offset is at the center of the image: $p_x = \frac{W}{2}$, $p_y = \frac{H}{2}$. An important case where it is far off from the image center is when an image has been cropped asymmetrically.

Pixel Aspect Ratio: Sensors that do not have square pixels, *i.e.*, sensors that have a different number of pixels when going one unit length in x- and y-direction, respectively, can be modeled through a factor a in the second focal length:

$$\begin{pmatrix} f & 0 & p_x & 0 \\ 0 & af & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} fx + p_x z \\ afy + p_y z \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{fx}{z} + p_x \\ \frac{afy}{z} + p_y \end{pmatrix} = \mathbf{i}. \tag{2.7}$$

In most consumer cameras this factor is, however, very close to 1 (and thus not estimated by some calibration or structure from motion frameworks). The matrix

$$\mathbf{K} = \begin{pmatrix} f & 0 & p_x \\ 0 & af & p_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

is called *calibration matrix* and f, a, p_x, and p_y are called *intrinsic parameters* or simply *intrinsics*.

2.1.1.2. Rotation and Translation

Earlier we made the restriction that the camera must be positioned at the world coordinate frame's origin looking into the world's z-direction. If a point is given in world coordinates and the camera has its own *camera coordinate frame* not fulfilling this restriction, we must rotate and translate the world coordinate frame such that it coincides with the camera coordinate frame before the actual projection. With $\mathbf{R}$ being a suitable rotation matrix and $\mathbf{c}$ being the position of the camera center in world coordinates, a point $\tilde{\mathbf{x}}_{\text{world}}$ in world coordinates is transformed into the camera coordinate frame as follows:

$$\tilde{\mathbf{x}}_{\text{camera}} = \begin{pmatrix} \mathbf{R} & -\mathbf{Rc} \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix} \tilde{\mathbf{x}}_{\text{world}}. \tag{2.9}$$

$\mathbf{R}$ and $\mathbf{c}$ are called *extrinsic parameters* or simply *extrinsics*.

Now we have all parts together to describe the complete transformation $\mathbf{P}\tilde{\mathbf{x}}$ of a point $\tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ in the world into pixel coordinates:

$$\begin{aligned} \mathbf{P}\tilde{\mathbf{x}} &= \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & af & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{R} & -\mathbf{Rc} \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & af & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} (\mathbf{R})_{1,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{1,\cdot} \\ (\mathbf{R})_{2,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{2,\cdot} \\ (\mathbf{R})_{3,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{3,\cdot} \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} f\big((\mathbf{R})_{1,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{1,\cdot}\big) + p_x\big((\mathbf{R})_{3,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{3,\cdot}\big) \\ af\big((\mathbf{R})_{2,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{2,\cdot}\big) + p_y\big((\mathbf{R})_{3,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{3,\cdot}\big) \\ (\mathbf{R})_{3,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{3,\cdot} \end{pmatrix} \\ &\mapsto \begin{pmatrix} f\frac{(\mathbf{R})_{1,\cdot}\mathbf{x} - (\mathbf{Rc})_{1,\cdot}}{(\mathbf{R})_{3,\cdot}\mathbf{x} - (\mathbf{Rc})_{3,\cdot}} + p_x \\ af\frac{(\mathbf{R})_{2,\cdot}\mathbf{x} - (\mathbf{Rc})_{2,\cdot}}{(\mathbf{R})_{3,\cdot}\mathbf{x} - (\mathbf{Rc})_{3,\cdot}} + p_y \end{pmatrix} = \mathbf{i}, \end{aligned} \tag{2.10}$$

with $(\mathbf{A})_{i,\cdot}$ being the i-th row of a matrix or vector $\mathbf{A}$. One often refers to an image's matrix $\mathbf{P} = \mathbf{K}\begin{pmatrix} \mathbf{R} & -\mathbf{Rc} \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix}$ as this image's *camera* or *view*, for example: "Cameras 1

and 2 are identical except for a horizontal translation." Thus, two images can have different "cameras" even if they were shot with the same physical camera.

2.1.1.3. Lenses and Distortion

The pinhole camera is just a simplified mathematical model. For maximal image sharpness the pinhole should, in theory, be infinitesimal because a ray can then only enter the camera through one clearly defined point and consequently only hits the image plane at a single point, leading to a very sharp image. In practice, however, a very small pinhole lets very little energy enter the camera and the exposure time has to be extensive. Another problem is that very small pinholes introduce light diffraction effects that deteriorate the image quality. In practice one therefore uses lenses which approximate a pinhole camera and let more light enter the camera. However, they do not behave exactly like pinhole cameras, the most important and prominent difference being that not every point in the scene appears sharp in the image. Points that fulfill the following equation (for thin lenses) are being projected to a single point and appear sharp:

$$\frac{1}{\text{distance between lens and object}} + \frac{1}{\text{distance between lens and image}} = \frac{1}{f}. \quad (2.11)$$

Here the *focal length* f differs from the definition in the pinhole camera model:

$$\begin{aligned} \frac{1}{f} &\approx (n-1)\left(\frac{1}{R_1} - \frac{1}{R_2}\right) \\ &\neq \frac{1}{\text{distance between lens and image}}, \end{aligned} \quad (2.12)$$

where R_1 and R_2 are the radii of curvature of the two lens surfaces and n is the lens material's refraction index. All other points get spread out into a so-called circle of confusion and if this circle of confusion is larger than the image's spatial sampling rate (*i.e.*, the pixel diameter), they appear blurred. Apart from blurring, this effect does not modify the geometric configuration of points.

What does, however, modify the geometric configuration, is lens distortion. Lens distortion means that world points are not projected on the image where they should be according to the above equations, but closer to or further away from the principal point. As a result, straight lines in the world appear bent in the image. The two most common kinds of distortion are so-called pincushion distortion (see Figure 2.2 (left)) and barrel distortion (see Figure 2.2 (center and right)).

The strength of the distortion depends on an image point's radial distance r to the principal point – at least for the following model. Here we follow Szeliski [2010, pp. 52f] but with a slightly different notation: Let $\binom{x_{\text{before}}}{y_{\text{before}}}$ be a point's pixel coordinates after division by $(\mathbf{R})_{3,\cdot}\,\mathbf{x} - (\mathbf{Rc})_{3,\cdot}$ but before scaling by f and shifting

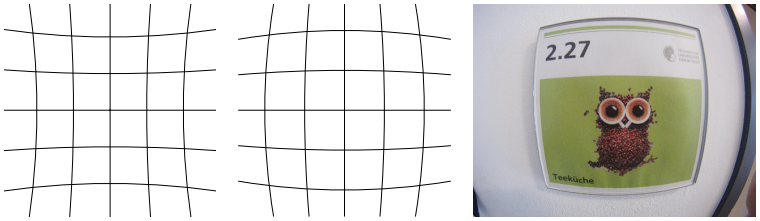


Figure 2.2.: *Left to right:* Pincushion distortion, barrel distortion, and a practical example of a quadratic door plate captured with a wide-angle camera. The first and second image were taken and modified from Wikimedia Commons (both by user WolfWings, public domain, `https://commons.wikimedia.org/wiki/File:Barrel_distortion.svg` and `https://commons.wikimedia.org/wiki/File:Pincushion_distortion.svg`).

by $\binom{p_x}{p_y}$, *i.e.*,

$$\begin{pmatrix} x_{\text{before}} \\ y_{\text{before}} \end{pmatrix} = \begin{pmatrix} \frac{(\mathbf{R})_{1,\cdot}\mathbf{x}-(\mathbf{Rc})_{1,\cdot}}{(\mathbf{R})_{3,\cdot}\mathbf{x}-(\mathbf{Rc})_{3,\cdot}} \\ \frac{(\mathbf{R})_{2,\cdot}\mathbf{x}-(\mathbf{Rc})_{2,\cdot}}{(\mathbf{R})_{3,\cdot}\mathbf{x}-(\mathbf{Rc})_{3,\cdot}} \end{pmatrix}. \tag{2.13}$$

The distorted coordinates are then

$$\begin{pmatrix} x_{\text{distorted}} \\ y_{\text{distorted}} \end{pmatrix} = \left(1 + \kappa_1 r^2 + \kappa_2 r^4\right) \begin{pmatrix} x_{\text{before}} \\ y_{\text{before}} \end{pmatrix}, \tag{2.14}$$

with $r^2 = x_{\text{before}}^2 + y_{\text{before}}^2$ and some *radial distortion parameters* κ_1 and κ_2. We obtain the final pixel coordinates via

$$\begin{pmatrix} x_{\text{final}} \\ y_{\text{final}} \end{pmatrix} = \begin{pmatrix} f x_{\text{distorted}} + p_x \\ a f y_{\text{distorted}} + p_y \end{pmatrix}. \tag{2.15}$$

According to Szeliski [2010] this model does not hold, for instance, for fisheye lenses, but it produces good results for most consumer lenses.

Even though distortions are a nuisance because they cannot be expressed as linear operations like the rotation, translation, and projection earlier, the fact that the distortion only depends on the distance to the principal point (and not, say, its position in the world) means that we can estimate the radial distortion parameters κ_1 and κ_2 once (*e.g.*, with calibration targets or structure from motion (Section 2.2.1)), remove the distortion from an image, and use the undistorted image and linear operations in all following steps.

2.1.2. Epipolar Geometry

We briefly introduce epipolar geometry here, since it follows from the projective geometry described above and is fundamental for structure from motion (Section 2.2.1), stereo (Section 2.2.2), and image-based rendering (Section 2.3).

Let us assume we have a configuration as depicted in Figure 2.3 where two cameras look at the same scene but from different viewpoints. We pick a point $\mathbf{i}_l$ in the left

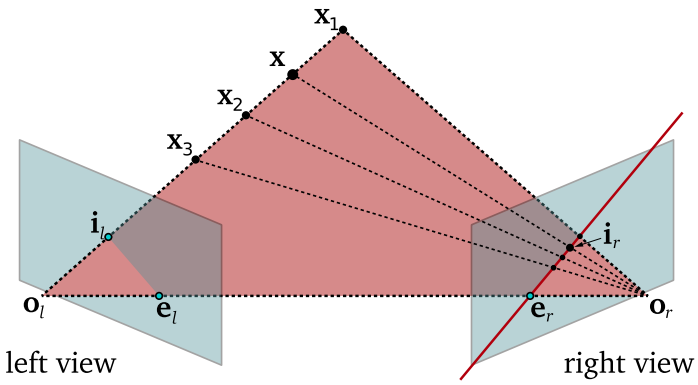


Figure 2.3.: Illustration of epipolar geometry. Two cameras with their camera centers $\mathbf{o}_l$ and $\mathbf{o}_r$ are viewing the same scene. We are interested in the true 3D position $\mathbf{x}$ of image point $\mathbf{i}_l$ which can be anywhere on the line $\overline{\mathbf{o}_l\mathbf{i}_l}$. When projecting all hypotheses ($\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, ...) for $\mathbf{x}$ into the right view, they form a line (red) called the epipolar line.

image and we are interested in its 3D position $\mathbf{x}$ in the scene. It can only be on the *viewing ray* $\overline{\mathbf{o}_l\mathbf{i}_l}$ from the left camera center $\mathbf{o}_l$ through the (3D position of the) pixel $\mathbf{i}_l$. Its distance from the camera – its *depth*[11] – is, however, unknown. Candidates for the true position are for example $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$. If we take all hypotheses, *i.e.*, the complete ray $\overline{\mathbf{o}_l\mathbf{i}_l}$, and project them into the right view, we obtain the red line called the *epipolar line*. The hypothesis with a depth of 0, *i.e.*, the left camera center $\mathbf{o}_l$, projects onto the point $\mathbf{e}_r$ which is called right *epipole*. The (red) plane that spans between $\overline{\mathbf{o}_l\mathbf{o}_r}$ and $\overline{\mathbf{o}_l\mathbf{i}_l}$ is called the *epipolar plane*. In order to find the true projection $\mathbf{i}_r$ of $\mathbf{x}$ in the right image, we only need to search on the epipolar line.

We call the relationship between two image points $\mathbf{i}_l$ and $\mathbf{i}_r$ that are both images of the same world point $\mathbf{x}$ a *point correspondence* – dense correspondence if we are interested in all points of an image[12] and sparse correspondence if we are only interested in certain points. The purpose of binocular (*i.e.*, two cameras) or multi-view (*i.e.*, more than two cameras) stereo algorithms is precisely to establish dense correspondence between images given extrinsic and intrinsic camera parameters. Here, the restriction of correspondence search to epipolar lines fundamentally limits the search space compared to optical flow, where the whole second image has to be searched for correspondences. Apart from stereo, point correspondences are also used in image-based "rendering with implicit geometry"[13] in the image-based rendering continuum of Kang et al. [2006] that we explain on page 30.

[11] Following the convention of MVE [Fuhrmann et al. 2014, 2015], Hartley and Zisserman [2004], and many other works, we will in this thesis use a point's distance from the camera center *measured along the principal axis* (as opposed to radial distance) as depth. Later we will also use the concept of *depth maps* – images in which each pixel's value denotes its depth.

[12] except for those that are visible in one but not in the other image due to occlusions

[13] With implicit geometry Kang et al. [2006] mean point correspondences rather than explicitly computed geometry.

2.1.2.1. The Fundamental Matrix

Let us now take a closer look at how we can use epipolar geometry mathematically. The position of the epipolar line depends on the position of $\mathbf{i}_l$ and the extrinsic and intrinsic parameters of both cameras. Let $\tilde{\mathbf{i}}_l$ and $\tilde{\mathbf{i}}_r$ be a pair of corresponding points (*i.e.*, the projections of a 3D point $\mathbf{x}$ in the left and right view) in homogeneous 2D coordinates in their respective image coordinate frames, *i.e.*, $\tilde{\mathbf{i}}_l = \begin{pmatrix} x_l \\ y_l \\ 1 \end{pmatrix}$ and $\tilde{\mathbf{i}}_r = \begin{pmatrix} x_r \\ y_r \\ 1 \end{pmatrix}$. Then there exists a 3×3 matrix $\mathbf{F}$ such that

$$\tilde{\mathbf{i}}_r^\mathsf{T} \mathbf{F} \tilde{\mathbf{i}}_l = 0 \tag{2.16}$$

for all corresponding point pairs $(\tilde{\mathbf{i}}_l, \tilde{\mathbf{i}}_r)$ [Faugeras 1992]. The matrix $\mathbf{F}$ is called the *fundamental matrix* and Equation (2.16) is called the correspondence condition [Hartley and Zisserman 2004, Section 9.2.3]. $\mathbf{F}\tilde{\mathbf{i}}_l$ is the epipolar line (in normal form in homogeneous coordinates) of $\mathbf{i}_l$ in the right view. Similarly, $\tilde{\mathbf{i}}_r^\mathsf{T}\mathbf{F}$ (or $\mathbf{F}^\mathsf{T}\tilde{\mathbf{i}}_r$) is the epipolar line of $\mathbf{i}_r$ in the left view. We are not going to go into details regarding the properties of the fundamental matrix and rather refer the reader to Hartley and Zisserman [2004, Section 9.2.4]. What matters for our purpose is the following.

The fundamental matrix can be computed from the two cameras' calibration matrices: Let $\mathbf{P}_l = \mathbf{K}_l \cdot \begin{pmatrix} \mathbf{R}_l & \mathbf{t}_l \\ \mathbf{0}_{1\times3} & 1 \end{pmatrix}$ be the left and $\mathbf{P}_r = \mathbf{K}_r \cdot \begin{pmatrix} \mathbf{R}_r & \mathbf{t}_r \\ \mathbf{0}_{1\times3} & 1 \end{pmatrix}$ be the right camera matrix, and $\mathbf{A}^+$ be the pseudo-inverse of $\mathbf{A}$. The fundamental matrix can then be written [Hartley and Zisserman 2004, Equation 9.2] as

$$\mathbf{F} = [\mathbf{P}_r \mathbf{o}_l]_\times \mathbf{P}_r \mathbf{P}_l^+. \tag{2.17}$$

After computing $\mathbf{F}$, it could, for example, be used to restrict the stereo correspondence search space to the epipolar lines. In 3D reconstruction or more specifically structure from motion (Section 2.2.1) we are, however, not interested in computing $\mathbf{F}$ from camera matrices but the inverse: We search for point correspondence candidates in a pair of images (without the aid of the fundamental matrix), estimate the fundamental matrix from those candidates [Hartley 1997], and decompose $\mathbf{F}$ into camera matrices.

2.1.2.2. Triangulation

After we have established a point correspondence such as $(\mathbf{i}_l, \mathbf{i}_r)$ in Figure 2.3, we can check where the viewing rays $\overline{\mathbf{o}_l\mathbf{i}_l}$ and $\overline{\mathbf{o}_r\mathbf{i}_r}$ intersect to obtain the true 3D position of $\mathbf{x}$. This process is called triangulation. In practice, triangulation is not as simple: Due to camera parameter inaccuracies, rounding, and numerical inaccuracies the rays will in practice pass each other with a non-zero distance. One then searches for the 3D point with minimal distance to both rays, which is the center of a line segment orthogonal to $\overline{\mathbf{o}_l\mathbf{i}_l}$ and $\overline{\mathbf{o}_r\mathbf{i}_r}$. When working with multiple cameras (*i.e.*, multi-view stereo), we have multiple lines for triangulation and seek the least-squares solution that minimizes the distance to all these lines. Here, the multiple cameras help improve the triangulation's accuracy.

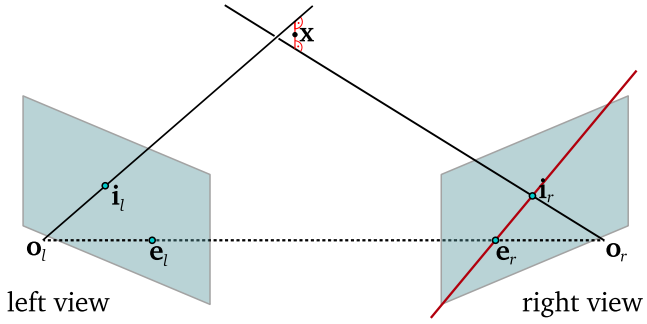


Figure 2.4.: Given a point correspondence $(\mathbf{i}_l, \mathbf{i}_r)$ the viewing rays $\overline{\mathbf{o}_l, \mathbf{i}_l}$ and $\overline{\mathbf{o}_r, \mathbf{i}_r}$ generally do not intersect in 3D and we have to find a point $\mathbf{x}$ that minimizes the distance to all rays. For two rays this is the center of the red line segment orthogonal to both rays.

2.1.2.3. Inverse Projection and Reprojection to Another View

Earlier we explained how a world point $\mathbf{x}$ is projected onto a homogeneous 2D image point $\tilde{\mathbf{i}}$. We can also invert this process – *i.e.*, project the 2D point back into the world – if we know the point's depth [Hartley and Zisserman 2004, p. 250]: Let $\mathbf{P} = \mathbf{K} \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix}$ be the camera matrix and d be point $\mathbf{x}$'s depth (*i.e.*, the distance from the camera measured along the principal axis). Then

$$\mathbf{x} = \mathbf{R}^{-1}\left(\mathbf{K}^{-1}\tilde{\mathbf{i}}\, d - \mathbf{t}\right) = \mathbf{R}^{\mathsf{T}}\left(\mathbf{K}^{-1}\tilde{\mathbf{i}}\, d - \mathbf{t}\right) \tag{2.18}$$

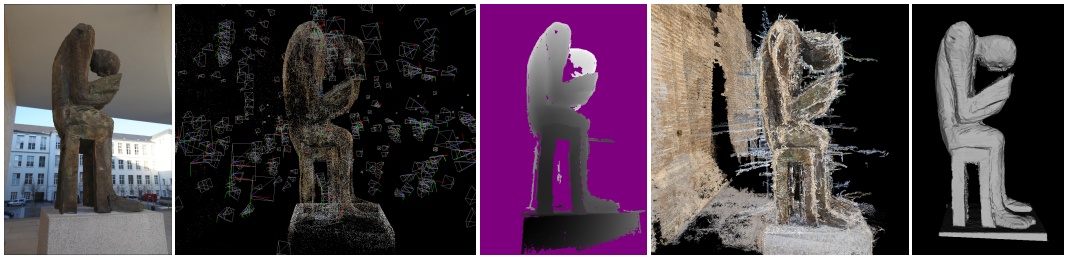
$$\text{with } \mathbf{K}^{-1} = \begin{pmatrix} \frac{1}{f} & 0 & -\frac{p_x}{f} \\ 0 & \frac{1}{af} & -\frac{p_y}{af} \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.19}$$

Instead of only projecting an image point back into the world, we can also project it into the world *and* then reproject it into a novel view:

$$\tilde{\mathbf{i}}_{\text{novel}} = \mathbf{K}_{\text{novel}}\left(\mathbf{R}_{\text{novel}}\mathbf{R}^{\mathsf{T}}\left(\mathbf{K}^{-1}\tilde{\mathbf{i}}\, d - \mathbf{t}\right) + \mathbf{t}_{\text{novel}}\right). \tag{2.20}$$

2.2. Image-Based Modeling / 3D Reconstruction

Although this thesis does not tackle the problem of geometry reconstruction from images, it has many intersection points with this field: Part I of this thesis uses reconstructed 3D models and calibrated images as input and Part II evaluates IBR systems that require camera parameters and reconstructed 3D models. Camera parameters can be obtained with structure from motion if we want to work on general images without any calibration targets, and 3D models can be obtained with multi-view stereo followed by surface reconstruction. To make the basic concepts of 3D reconstruction clear to the reader, this section gives an overview over how a basic pipeline works. We will mainly follow the workflow of the pipeline MVE [Fuhrmann et al. 2014, 2015] and give pointers in other directions here and there.



(a) (b) (c) (d) (e)

Figure 2.5.: Complete image-based 3D reconstruction pipeline on a dataset of the statue "Der Lesende" (The Reader). **(a)** One of 540 input photos, **(b)** structure from motion result (points represent triangulated point correspondences and gray pyramids represent cameras), **(c)** depth map from multi-view stereo where pixel colors represent depth (white=distant, black=close, magenta=no depth), **(d)** all depth maps reprojected into 3D space (with many outliers), and **(e)** the result of surface reconstruction.

We start with structure from motion (Section 2.2.1), which calibrates (*i.e.*, computes camera parameters for) an image collection from nothing but the image data and the focal lengths in the images' EXIF tags. We then explain in Section 2.2.2 how, given such calibration data, multi-view stereo computes dense 3D point clouds. Finally, in Section 2.2.3 we give a brief introduction into how dense 3D point clouds can be turned into surface meshes. These steps are visualized in Figure 2.5.

2.2.1. Structure from Motion

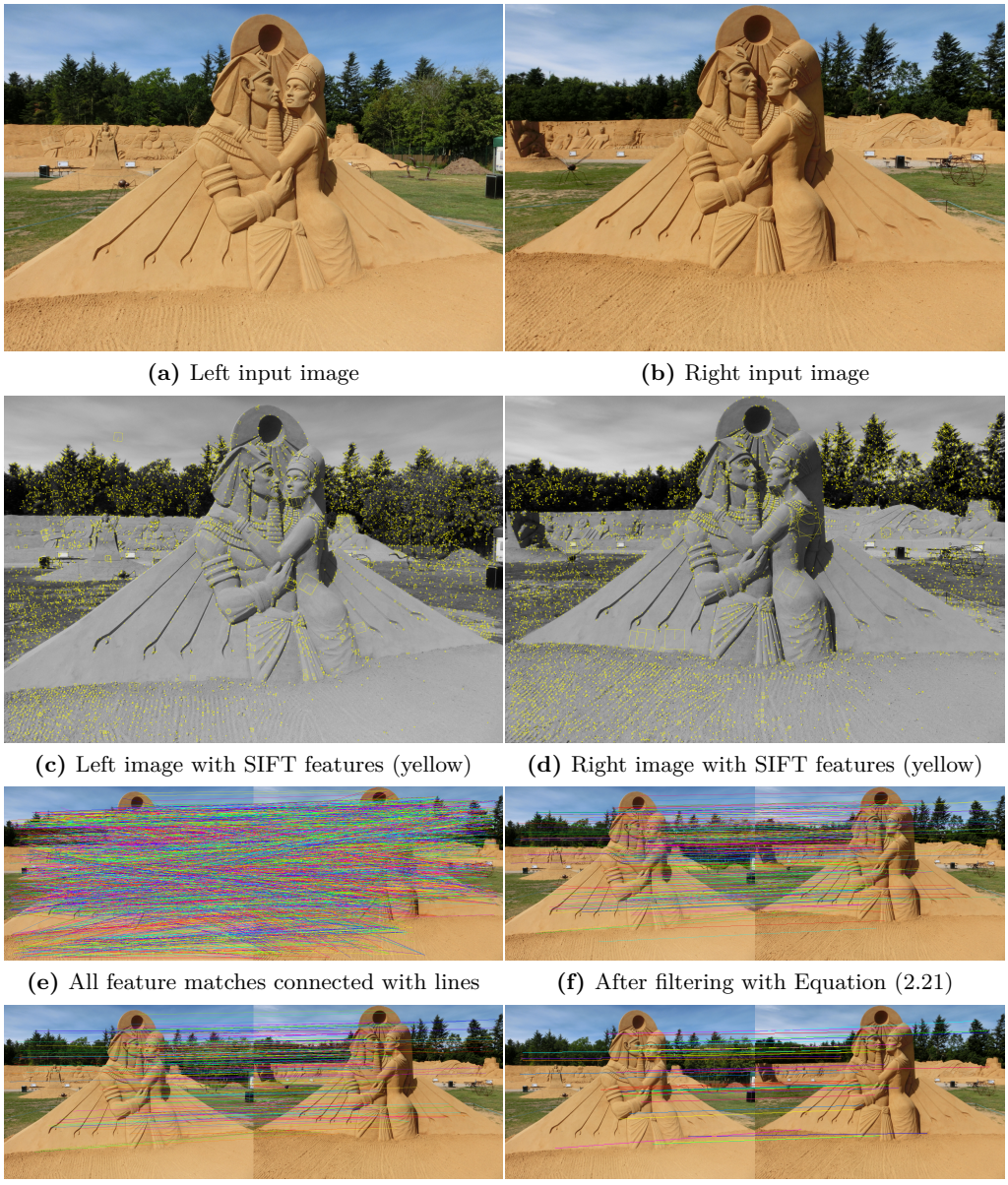
Given a collection of photos, structure from motion[14] performs *image matching*, *i.e.*, it tries to find out which of the images partially show the same scene content. The first work to do this robustly on unstructured collections of tourist photos was the seminal "Photo Tourism" paper [Snavely et al. 2006].

The first step of structure from motion is finding point correspondences between images. We already scratched the surface of this in Section 2.1.2.1. Here, point correspondences have to be found without any geometric or camera information; only the image content is available at this point. Correspondences must fulfill Equation (2.16) but with an unknown fundamental matrix. After obtaining point correspondences, we can compute the fundamental matrix and decompose it into camera matrices.

2.2.1.1. Feature Matching

So, how do we establish point correspondences only based on image content? First we determine points such as corners [Harris and Stephens 1988] or blobs in each image that are distinctive. This is called a *feature detector* and the points are *features*. Such features can be seen as yellow squares in Figures 2.6c and 2.6d. We

[14] The "motion" refers to motion of the camera, not the observed object. According to Soatto (stefano.soatto.sity.com/ucla-computer-science-department) "the ability to move is fundamental to intelligence." Here it is fundamental for reasoning about 3D geometric relations in the world.



(a) Left input image **(b)** Right input image

(c) Left image with SIFT features (yellow) **(d)** Right image with SIFT features (yellow)

(e) All feature matches connected with lines **(f)** After filtering with Equation (2.21)

(g) After filtering with mutuality constraint **(h)** After geometric verification

Figure 2.6.: Feature detection and matching and match filtering on a sand art piece.

then need to find a representation for a feature and its local image neighborhood such that we can compare it with features in other images and find the same scene region again based on this representation. This is called a *feature descriptor*.

An early descriptor was SIFT [Lowe 1999, 2004] which was better than most competitors at the time of its publication [Mikolajczyk and Schmid 2005]. It was followed by a plethora of other descriptors such as SURF [Bay et al. 2006], the motivation behind many of them being the reduction of memory consumption or

speeding up the feature computation or matching. Even today, SIFT still performs respectably despite its age [Miksik and Mikolajczyk 2012, Table 4]. For the purpose of this thesis we do not need to go into details regarding SIFT's internals and rather treat it as a black box.

SIFT turns a feature point and its neighborhood into a *feature vector* of 128 bytes. With this descriptor we can then do a nearest neighbor search on the descriptors of features in other images to find the descriptor with the smallest Euclidean distance – called a *feature match* – which describes the same 3D point in the scene captured from a different viewpoint (unless it is a false match of course). Such feature matches are shown in Figure 2.6e, which does, however, contain many false matches.

An important concept in the context of descriptors is invariance: SIFT's detector finds more or less the same detections under changes in scale, *e.g.*, after zooming out of a scene or downscaling an image. Further, SIFT's descriptor does not change if the camera is rotated around the principal axis and it does not change much under affine scene transformations (scaling, rotation, shearing). As a result, two cameras looking at a scene point with viewing angles that differ (viewing angle difference is called *parallax angle*) by less than 50° can still be matched. Finally, SIFT is robust against photometric differences due to changes in illumination or exposure.

Because SIFT detects many false matches, there are various ways to filter them out. The first two are only based on descriptor content, not on geometric reasoning. First, when searching for a query feature's nearest neighbor, nearest neighbors that do not match much better than the second nearest neighbor are filtered out, which is expressed in the following equation:

$$\frac{\|\text{SIFT(nearest neighbor)} - \text{SIFT(query)}\|_2}{\|\text{SIFT(second nearest neighbor)} - \text{SIFT(query)}\|_2} < t_1 \text{ (\textit{e.g.}, } t_1 = 0.8). \quad (2.21)$$

The result of this filtering process can be seen in Figure 2.6f. Second, one frequently requires a match to be mutual, *i.e.*, we filter out matches where feature $\mathbf{f}_1$ is the nearest neighbor of feature $\mathbf{f}_2$ but $\mathbf{f}_2$ is not the nearest neighbor of $\mathbf{f}_1$. The remaining features (see Figure 2.6g) contain much fewer false correspondences.

2.2.1.2. Geometric Verification

In the next step we take an image pair and compute a fundamental matrix from the filtered matches. This can be done based on eight point correspondences [Hartley 1997], but since the filtered matches still contain an unknown number of false matches, this eight point algorithm is embedded in a robust framework called RANSAC [Fischler and Bolles 1981]: We pick eight random matches, compute a hypothesis for the fundamental matrix, check how many so-called *inliers* this hypothesis has, *i.e.*, how many of the SIFT matches approximately fulfill the correspondence condition (Equation (2.16) but slightly relaxed to $|\tilde{\mathbf{i}}_r^\top \mathbf{F} \tilde{\mathbf{i}}_l| \leq t_2$ for some error threshold t_2) with this hypothesis matrix, and then we repeat this procedure for a predetermined number of iterations. Afterwards we keep the hypothesis with the highest number of inliers and may optionally refine it by estimating the fundamental matrix with all

inliers instead of only eight correspondences. In Figure 2.6h we show all inliers for the best fundamental matrix of our example image pair. If the fundamental matrix that we found has fewer than a certain number of inliers (*e.g.*, 20 [Snavely et al. 2006]), the feature matches and the fundamental matrix are rejected and the image pair regarded as not matching.

To find all matching image pairs we can compare all $\frac{N}{2}(N-1)$ image pairs. Unfortunately, this is prohibitively expensive on datasets with more than 10,000 images. Many methods have been proposed to reduce the complexity of this process, the most important methods [Agarwal et al. 2009, Frahm et al. 2010, Havlena and Schindler 2014, Heinly et al. 2015] being based on vocabulary trees [Nistér and Stewénius 2006], but all of these miss some image matches that could be found with exhaustive image matching.

The matched and verified feature pairs are now combined in a transitive manner: If feature $\mathbf{f}_1$ in image I_1 matches $\mathbf{f}_2$ in I_2 and $\mathbf{f}_2$ matches $\mathbf{f}_3$ in I_3, all three features are associated with each other and regarded as showing the same scene point. This is called a *feature track*. Next we need to filter out contradictions: If we can transitively follow $\mathbf{f}_1$ in image I_1 through several other images and then back into I_1 and there it matches a different feature than $\mathbf{f}_1$, this is a contradiction because in I_1 no other feature can correspond to the same scene point as $\mathbf{f}_1$. Snavely et al. [2006, Section 4.1] solve this by simply discarding such inconsistent tracks. All remaining feature tracks will later be triangulated into sparse scene points such as the points in Figure 2.5b.

2.2.1.3. Incremental Structure from Motion

Given the geometrically verified feature correspondences from above, in incremental structure from motion one then starts with one matched image pair, infers the camera parameters for it, triangulates all tracks in this pair into 3D points, refines camera parameters and 3D point positions through a process called *bundle adjustment*, adds another view that has considerable overlap with the initial view pair, triangulates tracks again, performs bundle adjustment again, and repeats this process until no further image can be added.[15]

The initial image pair that is used to start the incremental structure from motion is selected among all matched image pairs subject to the following [Snavely et al. 2006]: It should have a large number of shared tracks while not having a negligible baseline and not having point correspondences mostly on a single plane. The latter two cases would later produce degenerate cases and can be avoided by checking whether more than half of all point correspondences can be explained by a homography.

[15] In contrast to incremental SfM there is also global SfM that tries to infer cameras jointly for all images. This can, for example, be done with a Markov random field formulation [Crandall et al. 2013] where MRF labels correspond to camera rotations and translations (see Section 2.4 for details on MRF optimization). Global SfM can in principle solve incremental SfM's problems with camera drift and loop closing but results of current systems are still mixed [Knapitsch et al. 2017].

Inferring Camera Matrices for the Initial Image Pair: After such an initial pair has been found, we initialize both cameras' calibration matrices $\mathbf{K}_l$ and $\mathbf{K}_r$ by setting the principal points to the image centers and the focal lengths to the values provided in the images' EXIF tags,[16] set the left camera's external parameters to $\mathbf{R}_l = \mathbf{I}_3$ and $\mathbf{t}_l = \mathbf{0}_{3\times1}$,[17] convert the fundamental matrix into the so-called *essential matrix* $\mathbf{E} = \mathbf{K}_r^\mathsf{T}\mathbf{F}\mathbf{K}_l$ [Hartley and Zisserman 2004, Equation (9.12)], and compute the external parameters $\mathbf{R}_r$ and $\mathbf{t}_r$ of the right camera from $\mathbf{E}$ as described by Hartley and Zisserman [2004, Sections 9.6.2 and 9.6.3].

Triangulation: Using the obtained camera parameters, we can then triangulate all tracks shared between the initial image pair as sketched in Section 2.1.2.2 or described in detail by Szeliski [2010, Section 7.1].

Bundle Adjustment: We now have sparse 3D scene points $\tilde{\mathbf{x}}_m$ from the triangulated tracks and for each image I_n associated with this track we have a corresponding feature point $\tilde{\mathbf{i}}_{m,n}$. As shown in Figure 2.4 multiple corresponding viewing rays generally do not intersect in 3D and their triangulated track lies on none of the viewing rays. As a result the projection $\mathbf{P}_n\tilde{\mathbf{x}}_m$ of the triangulated track into I_n will not be identical with the position of the feature detection $\tilde{\mathbf{i}}_{m,n}$, *i.e.*, the *reprojection error*

$$\left\|\tilde{\mathbf{i}}_{m,n} - \mathbf{P}_n\tilde{\mathbf{x}}_m\right\|_2^2 \tag{2.22}$$

will be greater than zero due to inaccuracies in the estimated camera parameters, for example, because the focal lengths from the EXIF tags are inaccurate, there are non-zero image distortions, or the principal points are not at the image centers.

This problem is tackled in a non-linear least squares optimization procedure called *bundle adjustment* [Triggs et al. 1999] that takes all cameras' focal lengths, principal points, and distortion parameters and all 3D feature point positions as variables and minimizes the sum of all reprojection errors. To have a more stable optimization one may choose to exclude, for instance, the principal points from the optimization if they are known to be reasonably close to the image centers.

Adding Views: Next we add the image sharing the most features with the already triangulated tracks. We again initialize its calibration matrix with the EXIF focal length and the image center as principal point. To obtain the new image's external parameters we use correspondences between triangulated tracks and 2D features and put them into the perspective 3-point algorithm [Kneip et al. 2011].[18] We can

[16] Szeliski [2010, p. 313]: "In the absence of [additional] information, it is not possible to recover a [...] calibration matrix $\mathbf{K}_j$ for each image from correspondences alone."

[17] It is fundamentally impossible to recover the absolute scale and translation of a scene without additional information. If we know the length of certain objects in the scene such as a 2 m door frame, we can use this to obtain the scene's scale. Without such information we have to make certain assumptions, *e.g.*, $\mathbf{R}_l = \mathbf{I}_3$, $\mathbf{t}_l = \mathbf{0}_{3\times1}$, and $\mathbf{t}_r = (1, 0, 0)^\mathsf{T}$.

[18] In cases where an image has no EXIF tags it is also possible to recover the external parameters *and the focal length* from point correspondences [Snavely et al. 2006, Appendix A].

now triangulate tracks shared between the newly added image and the previous images and again refine camera parameters and 3D track positions through bundle adjustment. This process of adding an image, inferring its camera parameters, triangulating tracks, and running bundle adjustment is iterated until there remains no image to add. Due to the costliness of bundle adjustment one may choose to not run it after every added image but only after every k images. An exemplary result of structure from motion is shown in Figure 2.5b: The gray pyramids visualize camera positions, orientations, and fields of view and the points are the triangulated tracks.

2.2.2. Multi-View Stereo

For multiple images of a scene, structure from motion gives us camera parameters and a sparse reconstruction of the scene (the 3D feature points). The purpose of multi-view stereo is to take such calibration data and turn it into a *dense reconstruction*.

A very intuitive way to do this is to look at every image, segment it into foreground and background, regard the 3D space as voxel space, and "carve" away every voxel that projects into the background region of any image. This so-called *space carving* or visual hull carving [Potmesil 1987, Matusik et al. 2000, Kutulakos and Seitz 2000] has many downsides[19] and can today be regarded as obsolete.

The predominant line of research nowadays is to use the constraints of epipolar geometry covered in Section 2.1.2 with which we can infer the depth of almost any pixel corresponding to a surface point with Lambertian reflectance.[20]. Before we go into depth inference, we first need to briefly review another topic: view selection.

2.2.2.1. View Selection

When trying to infer the depth of pixels in an image – a so-called reference view – we need to select a set of so-called neighbor views (*e.g.*, 10 neighbors per reference [Goesele et al. 2007]) from all the images we have at hand. Those reference views should be the ones that "help us the most" with the depth inference task.

Obviously, we do not want images that show completely different scene parts. Goesele et al. [2007] therefore select neighbors that share as many features with the reference as possible. But because not all shared features have the same quality, they are weighted with two weighting functions. The first ensures scale compatibility, *i.e.*, it "favors views with equal or higher resolution than the reference view". The second looks at the parallax angle between reference and neighbor. Regarding this angle, there are two main effects: For too small angles, even just small matching

[19] The foreground/background segmentation may require user interaction and we cannot reconstruct the background, concavities, or object regions that none of the views observed from the side.

[20] Lambertian surfaces reflect light perfectly diffuse: A surface's apparent brightness is independent of the observer's viewing direction $\mathbf{v}$ onto the surface and follows a cosine law of incoming light $\mathbf{l}$ and surface normal $\mathbf{n}$: $I_d \propto \mathbf{l}^\mathsf{T}\mathbf{n} = \|\mathbf{l}\|_2 \|\mathbf{n}\|_2 \cos(\alpha)$ [Lambert and Anding 1892, §53]

inaccuracies lead to large depth inaccuracies. The angle that minimizes this inaccuracy problem is 90°. Unfortunately we cannot increase the angle to that point because too large angles decrease the matchability, *i.e.*, the ability to reliably find correct matches along the epipolar line as described in the next section. Therefore, the weighting function of Furukawa et al. [2010, Appendix A] (shown in Figure 2.7) increases up to an angle of 20° and then decreases again.

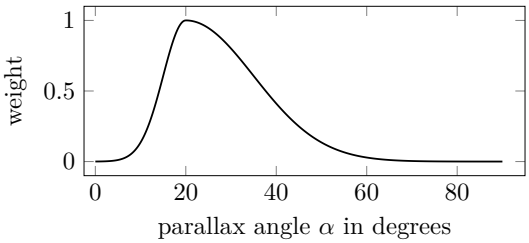


Figure 2.7.: The weighting function of Furukawa et al. [2010, Appendix A].

After selecting the set of neighbors subject to shared features, scale compatibility and parallax angle, all views are downscaled to the resolution of the view with the lowest resolution. The purpose of this is that (given that effects due to depth and surface orientation are already being accounted for in some other way) we need a surface area in the scene to span more or less the same number of pixels in all views.

For the time being, we will now only look at one reference image and one neighbor and look at multiple neighbors later. We will call them I_l and I_r independent of their true position to conform with the notation in Section 2.1.2. Given a reference and a neighbor, our task now is to infer depth according to the constraints of epipolar geometry: In Section 2.1.2 we already discussed that we can compute a pixel $\mathbf{i}_l$'s depth through finding its corresponding pixel $\mathbf{i}_r$ in the other view and triangulating the associated viewing rays. Further, $\mathbf{i}_r$ can only be found on the epipolar line $\mathbf{F}\tilde{\mathbf{i}}_l$ and a position on the epipolar line corresponds to a depth hypothesis. Thus, we can try out different depth hypotheses and check which one corresponds to an $\mathbf{i}_r$ that "looks most similar" to $\mathbf{i}_l$. This process is called matching and a function that measures similarity for that purpose is called a *matching cost function.*

2.2.2.2. Stereo Matching Costs

In the following, $\text{rep}_{l\to r}(\mathbf{i}, d)$ describes the reprojection of pixel $\mathbf{i}$ from image I_l into image I_r using a depth hypothesis d and both images' camera matrices (Section 2.1.2.3).

Given a pixel $\mathbf{i}_l$ in I_l and $\mathbf{i}_r = \text{rep}_{l\to r}(\mathbf{i}_l, d)$ in I_r, we now establish some measure of similarity between $\mathbf{i}_l$ and $\mathbf{i}_r$. Since single pixels are not very discriminative,[21]

[21] Discriminativeness here means that metrics should return very small differences if *and only if* $\mathbf{i}_l$ and $\mathbf{i}_r$ correspond to the same scene part. Depending on the application this should be invariant even if I_l and I_r were captured with different imaging modalities, but more on this later. Metrics operating on single pixels only cannot be invariant and discriminative at the same time. We will later see this with a single-pixel metric in our IBMR evaluation methodology (Chapter 7).

one typically considers a rectangular neighborhood patch $N_{\mathbf{i}_l}$ of size $m \times n$ around $\mathbf{i}_l$ and $N_{\mathbf{i}_l}$'s projection $N_{\mathbf{i}_r}$ in the right image. Given such a pair of rectangular patches,[22] there is a variety of metrics to compare their (dis)similarity. Since in most settings I_l and I_r are not captured with the same imaging modalities (*i.e.*, camera response curves, white balance, illumination, exposure, ISO), most of those metrics are to a varying degree designed to be invariant under changes in mean luminance or contrast.

We will now introduce a selection of popular metrics, partially following the naming scheme of Hirschmüller and Scharstein [2009]. We will present the metrics in a slightly more detailed manner than the rest of this chapter because some of them are used in our IBMR evaluation methodology in Part II of this thesis.

Sum of Squared Differences (SSD) is defined as

$$C_{\mathrm{SSD}}(\mathbf{i}_l, d; I_l, I_r) = \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} \left(I_l(\mathbf{i}) - I_r(\mathrm{rep}_{l \to r}(\mathbf{i}, d))\right)^2 . \tag{2.23}$$

If $N_{\mathbf{i}_r}$ shows the same image content as $N_{\mathbf{i}_l}$ but was for example exposed longer with all other settings being identical, $N_{\mathbf{i}_r}$ will be brighter than $N_{\mathbf{i}_l}$ and C_{SSD} will return a large difference even though $N_{\mathbf{i}_l}$ and $N_{\mathbf{i}_r}$ show essentially the same content. For the SIFT descriptor in Section 2.2.1 we already saw the concept of invariance under common imaging modality changes and we need to introduce this here as well. The first step is to remove each patch's mean as is done in the

Zero-Mean Sum of Squared Differences (ZSSD), which is defined as

$$C_{\mathrm{ZSSD}}(\mathbf{i}_l, d; I_l, I_r) = \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} \left((I_l(\mathbf{i}) - \mu_l) - (I_r(\mathrm{rep}_{l \to r}(\mathbf{i}, d)) - \mu_r)\right)^2 , \text{ with} \tag{2.24}$$

$$\mu_l = \frac{1}{|N_{\mathbf{i}_l}|} \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} I_l(\mathbf{i}) \text{ and } \mu_r = \frac{1}{|N_{\mathbf{i}_l}|} \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} I_r(\mathrm{rep}_{l \to r}(\mathbf{i}, d)).$$

Subtracting the patch means μ_l and μ_r does not, however, make ZSSD invariant under multiplicative changes.[23]

Normalized Cross-Correlation (NCC)[24] therefore normalizes with respect to the patches' means *and* standard deviations:

$$C_{\mathrm{NCC}}(\mathbf{i}_l, d; I_l, I_r) = \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} \frac{(I_l(\mathbf{i}) - \mu_l) \cdot (I_r(\mathrm{rep}_{l \to r}(\mathbf{i}, d)) - \mu_r)}{\sigma_l \sigma_r}, \text{ with} \tag{2.25}$$

[22] Even though the right patch is rarely rectangular in practice, we can think of both patches as being *implicitly* rectangular because we can reproject each pixel in $N_{\mathbf{i}_l}$ into I_r, look up I_r's color at that spot, and arrange the color lookups in a rectangular structure.

[23] At least in linear images, exposing I_r longer than I_l will make $N_{\mathbf{i}_r}$ look the same as $N_{\mathbf{i}_l}$ multiplied with a constant.

[24] Hirschmüller and Scharstein [2009] call this zero-mean NCC (ZNCC) which is somewhat inconsistent with the literature such as Szeliski [2010, Eq. 8.11].

μ_l and μ_r as above,

$$\sigma_l = \sqrt{\frac{1}{|N_{\mathbf{i}_l}| - 1} \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} (I_l(\mathbf{i}) - \mu_l)}, \text{ and}$$

$$\sigma_r = \sqrt{\frac{1}{|N_{\mathbf{i}_l}| - 1} \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} (I_r(\mathrm{rep}_{l \to r}(\mathbf{i}, d)) - \mu_r)}.$$

The $(|N_{\mathbf{i}_l}| - 1)^{-0.5}$ may be dropped from σ_l and σ_r because they only introduce a constant factor into C_{NCC} independent of the image content. Equation (2.25) is equivalent to computing (with $\mathbf{n}_p$ and $\mathbf{n}_q$ being the content of $N_{\mathbf{i}_l}$ and $N_{\mathbf{i}_r}$, respectively, stacked into column vectors)

$$C_{\mathrm{NCC}}(\mathbf{i}_l, d; I_l, I_r) = \frac{\mathbf{n}_p^\mathsf{T} \mathbf{n}_q}{\|\mathbf{n}_p\|_2 \, \|\mathbf{n}_q\|_2}. \tag{2.26}$$

Sinha et al. [2014, Section 5 and supplemental material] recommend adding a small constant to the denominator of Equation (2.25) so that the denominator does not reach zero for textureless image regions (*i.e.*, regions where $\sigma_l \approx 0$ and $\sigma_r \approx 0$):

$$C_{\mathrm{NCC}}(\mathbf{i}_l, d; I_l, I_r) = \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} \frac{(I_l(\mathbf{i}) - \mu_l) \cdot (I_r(\mathrm{rep}_{l \to r}(\mathbf{i}, d)) - \mu_r)}{\sqrt{\sigma_l^2 \sigma_r^2 + \varepsilon^2}}. \tag{2.27}$$

In contrast to (Z)SSD, NCC is a *similarity* metric, but we can turn it into a difference metric by using $1 - C_{\mathrm{NCC}}$ or $1 - \max(0, C_{\mathrm{NCC}})$ [Sinha et al. 2014, suppl. mat.].

A metric that is related to NCC in that it also normalizes with respect to patch mean and standard deviation is the

Structural Similarity (SSIM) Index which is defined as follows [Wang et al. 2004]:

$$C_{\mathrm{SSIM}}(\mathbf{i}_l, d; I_l, I_r) = \frac{(2\mu_l \mu_r + c_1)(2\sigma_{lr} + c_2)}{(\mu_l^2 + \mu_r^2 + c_1)(\sigma_l^2 + \sigma_r^2 + c_2)}, \text{ with} \tag{2.28}$$

$$\mu_l, \ \mu_r, \ \sigma_l, \text{ and } \sigma_r \text{ as above,}$$

$$\sigma_{lr} = \frac{1}{|N_{\mathbf{i}_l}| - 1} \sum_{\mathbf{i} \in N_{\mathbf{i}_l}} (I_l(\mathbf{i}) - \mu_l) \cdot (I_r(\mathrm{rep}_{l \to r}(\mathbf{i}, d)) - \mu_r),$$

$$c_1 = (0.01L)^2,$$

$$c_2 = (0.03L)^2, \text{ and}$$

$$L = \text{maximal luminance value } (L = 255 \text{ for } 8 \,\text{bit images}).$$

Like NCC it is also a similarity metric and we can turn it into a difference metric by using the *structural dissimilarity index* $\mathrm{DSSIM} = {}^{(1-\mathrm{SSIM})}/_{2}$.

Census is different from the previous metrics in that it does not perform arithmetic on the colors of $N_{\mathbf{i}_l}$ and $N_{\mathbf{i}_r}$, but only does a binary check on whether a pixel in a patch is lighter than the patch's center pixel [Zabih and Woodfill 1994]: It first stacks all luminances of a patch N into a vector and converts it into a binary descriptor $\mathbf{c}$ based on whether a pixel is brighter than the center pixel:

$$\mathbf{c}_l = \Big([I_l(\mathbf{i}) \geq I_l(\mathbf{i}_l)]\Big)_{\mathbf{i}\in N_{\mathbf{i}_l}} \text{ and } \mathbf{c}_r = \Big([I_r(\mathrm{rep}_{l\to r}(\mathbf{i}, d)) \geq I_r(\mathrm{rep}_{l\to r}(\mathbf{i}_l, d))]\Big)_{\mathbf{i}\in N_{\mathbf{i}_l}}, \tag{2.29}$$

with $[\,]$ being the Iverson bracket. In the second step these binary descriptors are then compared using the Hamming distance (number of different bits):

$$C_{\text{Census}}(\mathbf{i}_l, d; I_l, I_r) = \frac{1}{|N_{\mathbf{i}_l}|} \left\|\mathbf{c}_l - \mathbf{c}_r\right\|_{\text{Hamming}}. \tag{2.30}$$

Due to the binary expression $[\ldots \geq \ldots]$, Census is probably the most invariant among the metrics discussed here. The question is whether this invariance leads to a loss in discriminativeness, meaning that Census may evaluate many patches as being very similar even though they are not. In their matching cost evaluation, Hirschmüller and Scharstein [2009] found that this is not the case and Census "showed the best and most robust overall performance" – despite its simplicity.

Multi-View Matching Costs: We have so far only looked at pairwise stereo matching. In multi-view stereo we need to match one reference with multiple neighbors. The simplest way to do so is to regard the multi-view problem as a sum of two-view problems [Szeliski 2010, Equation 11.15], *i.e.*, we choose the depth that minimizes

$$C(\mathbf{i}_l, d; I_l) = \sum_{\substack{I_r \in \text{neigh-}\\ \text{bor views}}} C_{\text{two-view}}(\mathbf{i}_l, d; I_l, I_r). \tag{2.31}$$

2.2.2.3. Regularization

In practice, the depth with lowest matching costs is frequently not the actual correct depth. This is especially true if all candidate patches $N_{\mathbf{i}_r}$ along the epipolar line look very similar because the region around the epipolar line has little albedo variation (*e.g.*, a white wall in an indoor environment). A concept that helps us here is regularization: When looking at two neighboring pixels, their depth tends to be similar. Exceptions to this only occur at depth discontinuities, *i.e.*, where one pixel shows a foreground object and its neighbor shows another object further in the back. Depth discontinuities are relatively rare, otherwise the world would be a very rough, almost fractal place. We can see this in Figure 2.8 (*right*), where all depth changes (*i.e.*, gray value changes) happen gradually from one valid (*i.e.*, non-magenta) pixel to another except for the few regions where we have a foreground-background jump.

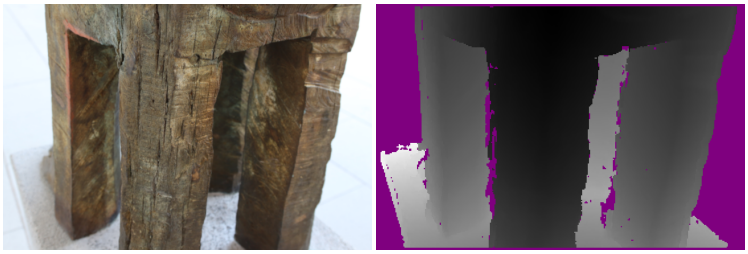


Figure 2.8.: Photo and reconstructed depth map (white: distant, black: close, magenta: no depth).

Thus, if we have two depth hypotheses d_1 and d_2 where d_1's matching costs are slightly inferior to d_2's, we may still prefer d_1 if it is significantly closer to the neighbor pixels' depth. We can enforce this so-called solution smoothness by giving penalties to pixels whose depth differs significantly from their neighbors and incorporate this in the objective function of global [Kolmogorov and Zabih 2002] or semi-global [Hirschmüller 2008, Bleyer et al. 2011] optimization methods.

With regularization we can go very far: A uniformly white wall leaves us completely clueless about its depth but we can infer every pixel's depth just from knowing the depth of the wall's four surrounding edges: It will be a smooth interpolation from the edges into the wall's center. We will encounter the concept of regularization again later, for example in our introduction to Markov random fields (Section 2.4).

At this point the reader should have a rough idea of how multi-view stereo works. In practice it is appreciably more involved: In addition to the view selection described above, Goesele et al. [2007] introduce *local view selection* that selects a different set of neighbor views for different pixels within a reference view. Further, we need to find and discard erroneous depths that result, *e.g.*, from occlusions and disocclusions at depth discontinuities, we need to find an optimization algorithm that performs matching cost and smoothness cost minimization efficiently, *etc.* But since the purpose of this background chapter is to give the reader a rough idea of the techniques used in image-based 3D reconstruction, we will leave it at that.

2.2.3. Surface Reconstruction

The result of multi-view stereo is a depth map[25] for every input image (see an example in Figure 2.5c). If we do 3D reconstruction in order to generate geometric proxies for image-based rendering (IBR, see the following section), depth maps may actually be sufficient because many IBR algorithms only require those (*e.g.*, Kopf et al. [2013]). For all others we can merge all depth maps into a global surface mesh using so-called *surface reconstruction*, see Figure 2.5e. In fact, even algorithms that only require depth maps may benefit from surface reconstruction because running surface reconstruction on multiple depth maps and then rendering the resulting surface into a depth buffer typically results in more accurate and complete depth

[25] Some algorithms output dense 3D point clouds [Furukawa and Ponce 2010] instead of depth maps.

maps than the direct multi-view stereo output.

Given a set of depth maps, we can take each pixel with a valid depth and project it back into 3D space (see Section 2.1.2.3). An example is shown in Figure 2.5d. Per se, each point in this point cloud is just a point and therefore infinitely small. However, we can associate a "size" with it: We can regard the pixel it came from as a square and check what extent the square's back projection has in the scene. This is called a *pixel footprint* or *pixel scale* [Fuhrmann and Goesele 2011, 2014]. In addition to scale, we can also assign an orientation to each point: Either the multi-view stereo algorithm directly provides an orientation for each pixel in the depth map, or we can derive it from the depth of the pixel and its neighbors.

Each of the resulting hundreds of millions of *depth samples* with their position, scale, and orientation are now an indication of where the scene's true surface is. They contain errors and noise, but we can use the fact that they are so plentiful to at least partially rule out both. Recent surface reconstruction algorithms [Mücke et al. 2011, Fuhrmann and Goesele 2011, 2014] do this as follows: They spread the information provided by a depth sample out in 3D space, *e.g.*, with 3D Gaussian functions [Mücke et al. 2011] (*i.e.*, the further away we are from a sample, the less likely we are to be on the true surface and this likelihood follows a Gaussian function on the distance to the sample) or with similar functions that are not rotationally symmetric [Fuhrmann and Goesele 2014]. The standard deviation of these functions is greater for samples with greater scale because such samples represent a larger region of the surface, but on the other hand they pinpoint the position of the surface less precisely.

These functions are then aggregated in 3D space for all depth samples. As a result, points in space where many samples provide evidence about the surface will have a very high function aggregate and vice versa. In the function aggregate, which is called the *implicit function*, it is important that the algorithm keeps a notion of whether a point is in front or behind the surface. Curless and Levoy [1996] and Fuhrmann and Goesele [2014] solve this with signed distance functions that indicate the distance to the surface and are negative behind and positive in front of a sample. This is why it is important to associate an orientation with each sample: Without the orientation, the signed distance function would indicate "in front" in some regions that are actually behind the true surface, and vice versa.

One then has to find the points where the implicit function transitions from "in front" to "behind", for example by finding the zero crossing in the aggregated signed distance functions [Curless and Levoy 1996, Fuhrmann and Goesele 2014]. This surface is then extracted and turned into a triangle mesh. The result needs to be cleaned up with respect to degenerate triangles, low-confidence mesh regions, and regions with very few triangles. An example output can be seen in Figure 2.5e.

Incorporating scale into surface reconstruction is important because in real-world datasets, the scale of images varies by several orders of magnitude: Consider, *e.g.*, an overview shot of Notre Dame Cathedral from a distance of several hundred meters and a close-up shot of a statue's face at Notre Dame's entrance portal. In such

a dataset we do not want the details provided by the close-up depth maps to be averaged out by the overview shots. Other current research directions include the reconstruction of surfaces with little support from depth samples [Jancosek and Pajdla 2011], thin objects [Ummenhofer and Brox 2013], or the incorporation of visibility constraints[26] [Jancosek and Pajdla 2011, Shan et al. 2014a].

2.3. Image-Based Rendering

As already mentioned, the goal of image-based rendering is to predict values of the plenoptic function at unknown points given a set of points with known values. More loosely speaking, given a set of images of a scene, we want to predict images of the same scene from novel viewpoints. In this section, we discuss various IBR algorithms from a high-level perspective, talk about their similarities and differences, and sound out their requirements and abilities.

Categorizing IBR Systems: Even though the plenoptic function (Equation (1.1)) is not a specific function or algorithm but rather an abstract goal definition for image-based modeling and rendering, it is still helpful for characterizing IBR algorithms by looking at what dimensions of the plenoptic function an algorithm does and does not model [Kang et al. 2006, Section 4.1, Table 4.1]. For example, a panoramic image mosaic [Szeliski and Shum 1997] can be seen as two-dimensional plenoptic function where the viewpoint (V_x, V_y, V_z) is fixed and we can only choose the angles (θ, ϕ), whereas light fields [Levoy and Hanrahan 1996] and the (not completely coincidentally similar) Lumigraph [Gortler et al. 1996] model a four-dimensional plenoptic function.

The number of dimensions that an algorithm models is related to

(a) how many samples we need to capture in order to reconstruct the plenoptic function well enough. We generally want to keep the setup as simple as possible and the number of samples to capture as low as possible.

(b) what liberties we have in placing the virtual camera. Kang et al. [2006] describe the goal as follows: "For [an IBR] representation to be compelling, it has to allow a reasonably wide range of viewpoints to be selected."

We will shed light onto these two points every now and then below.

Another system for characterizing IBR algorithms is what Kang et al. [2006] call the "IBR continuum". It orders IBR algorithms by the amount of geometry they require. Kang et al. chose the following IBR algorithms for illustration (some of them being historic and not too relevant anymore nowadays) and ordered them from those requiring less geometry to those requiring more geometry:

[26] Visibility constraints mean that we cannot only derive information from depth maps about where the surface should be but also about where it should not be. There should, *e.g.*, be no surface between a depth sample and the center of the camera it came from. If another depth map suggests that there is another depth sample within this space, it may be an error, especially if many depth maps suggest that this space should be empty.

Rendering with no geometry:

- Panorama mosaic [Szeliski and Shum 1997, Uyttendaele et al. 2001]
- Concentric mosaic [Shum and He 1999]
- Light field [Levoy and Hanrahan 1996]

Rendering with implicit geometry:

- View interpolation [Chen and Williams 1993]
- View morphing [Seitz and Dyer 1996]
- Joint view triangulation [Lhuillier and Quan 1999, 2003]
- Transfer methods [Laveau and Faugeras 1994, Avidan and Shashua 1997]

Rendering with explicit geometry:

- Layered depth images [Shade et al. 1998, Zitnick et al. 2004]
- Lumigraph [Gortler et al. 1996, Buehler et al. 2001]
- View-dependent geometry [Kang and Szeliski 2004]
- View-dependent texture [Debevec et al. 1996, Buehler et al. 2001, Eisemann et al. 2008]
- Static geometry, static texture (see, *e.g.*, Chapter 3)

At the top of this continuum, only pixel information is given. As discussed earlier, pixels correspond to rays (from the world through the camera center into the pixel). In the following we will therefore also refer to pixels in input images as *input rays* and pixels in virtual images as *virtual rays*. Without geometry information, algorithms do not know where a virtual ray hits the scene. But if we stay outside the convex hull of the objects of interest and roughly within the convex hull of the input cameras, it is possible to only operate in this so-called ray space, *i.e.*, take the vast collection of input rays and synthesize virtual rays from it by interpolating from the input rays closest to the virtual ray.

In the middle category, algorithms do not require geometry but sparse [Lhuillier and Quan 1999] or dense point correspondences [Chen and Williams 1993, Seitz and Dyer 1996, Laveau and Faugeras 1994]. Some approaches even work on image pairs with unknown internal parameters [Seitz and Dyer 1996], entirely uncalibrated pairs [Lhuillier and Quan 1999], or even without obeying epipolar geometry constraints [Lhuillier and Quan 1999]. We argue that the necessity of the implicit geometry category is debatable nowadays. On one hand, the well-founded theory of epipolar geometry [Hartley and Zisserman 2004] and the technology for calibrating even casually shot end-user photos [Snavely et al. 2006] do now exist and we do not need to refrain from using them. And on the other hand, it is only a tiny step (*i.e.*, triangulation) from dense correspondences to explicit geometry (given a static scene and calibrated cameras). Nevertheless, algorithms from this category may be helpful in specialized settings where it is, *e.g.*, known that the virtual camera only moves on a straight line [Seitz and Dyer 1996] or plane between input cameras.

At the bottom end of the continuum, explicit geometry information (called *impostor*, *geometric proxy* or simply *proxy*) is given. Proxies can be as simple as planes or more complex, such as planes with per-pixel depth variations ("sprite with depth"

or "plane plus parallax" [Shade et al. 1998]), depth maps, or global polygon meshes [Debevec et al. 1996, Eisemann et al. 2008]. Shade et al. [1998, Figure 1] point out that if a system allows the virtual camera to be close to the proxy, it should choose a proxy representation that allows for more geometric detail and vice versa – *i.e.*, planar proxies may be sufficient for objects far away from the virtual camera. The proxy may be modeled semi-manually[27] or with fully automatic image-based modeling techniques such as multi-view stereo and surface reconstruction. Given a proxy, IBR algorithms synthesize novel views by back-projecting the input images onto the proxy and then projecting them from there into the virtual camera. Back-projection and projection may be done in one single step (reprojection, see Section 2.1.2.3) at rendering time [Buehler et al. 2001] or the back-projection step may be done offline (traditional texture mapping) and the projection step at rendering time [Lempitsky and Ivanov 2007, Sinha et al. 2008, Garcia-Dorado et al. 2013]. Virtual pixels may be synthesized from one single input image [Lempitsky and Ivanov 2007] or from a weighted average of multiple input images [Debevec et al. 1996, Buehler et al. 2001, Callieri et al. 2008].

Selected IBR Systems: We now look at some specific IBR algorithms in a bit more detail: In panorama mosaicing [Szeliski and Shum 1997, Shum and Szeliski 1998, Uyttendaele et al. 2001] a user captures a handful of overlapping images with the camera rotating around its center. The algorithm then estimates two rotation angles for each image, reprojects the images into a spherical or cylindrical panorama, compensates for radiometric differences, and removes moving objects in the scene. This technique is advanced enough today that it works on photos that inexperienced users have casually captured with their phone cameras. It is debatable whether panorama mosaics are IBR algorithms according to the definition of McMillan and Bishop [1995] because they do not predict new points on the plenoptic function. The virtual camera can only be located precisely at the point that was captured by the input cameras.[28]

Concentric mosaics [Shum and He 1999] take the idea of panorama mosaics one step further: A camera is attached to an arm that rotates around a center point with the camera pointing in either tangential or normal direction. Images are then captured on multiple concentric circles while the arm rotates at constant angular speed. In this setup, the virtual camera can be rotated *and* translated within the

[27] There is a wealth of systems that create piecewise planar proxies from user annotations where users fit planes into the input images. See, *e.g.*, Debevec et al. [1996], Sinha et al. [2008], or Oliveira [2002, Section 4.1].

[28] A camera rotating around its own center has no stereo baseline which is a degenerate case for image-based modeling. It does not allow for depth inference and therefore every pixel in the panorama can be at any depth. Even moving the virtual camera just a tiny bit can, at least in theory, produce completely wrong results. If the camera is however rotated around some other point (see, *e.g.*, `https://developers.facebook.com/videos/f8-2017/casual-3d-capture`), we can do depth inference and image-based rendering with a virtual camera that is restricted to an area around the center of rotation.

capture circle. Unfortunately, concentric mosaics require this specialized rotating arm setup which make them somewhat specialized.

For a light field [Levoy and Hanrahan 1996] we need to capture images from camera positions on a regular, rectangular grid. Light fields share many properties with concentric mosaics (in fact, concentric mosaics can be seen as circularly acquired light fields): The acquisition process is a bit time-consuming,[29] the virtual camera can not only be rotated but also translated, and they assume that the light intensity does not change along a ray (*i.e.*, no occluders and no partially absorbing media such as smoke between the object of interest and any of the cameras). Further, they require the entire scene surface to be close to a so-called focal plane parallel to the input camera plane. All surface points far away from the focal plane will appear either aliased or blurry in the virtual views. The Lumigraph [Gortler et al. 1996] corrects this with the use of a proxy. It also requires the input images to be captured in a regular grid,[30] but this restriction was removed with the introduction of the unstructured Lumigraph [Buehler et al. 2001].

The unstructured Lumigraph is a generalization of light fields and view-dependent texturing, thereby forging a bridge across the IBR continuum. If its input cameras are placed on a grid and the proxy is a plane, it behaves like a light field, and if the cameras are placed irregularly and the proxy is a good approximation of the scene, it behaves like view-dependent texturing. View-dependent texturing works by choosing (and weighting) input images to be used as proxy texture depending on the position of the virtual camera. The power of view-dependent in contrast to static texturing is that it can give an illusion of fine geometric detail where there is none. In a city scene it can, *e.g.*, give the impression of windows being slightly recessed relative to their surrounding wall even if the entire wall including the windows is just modeled with one single plane in the proxy (see Debevec et al. [1996, Figures 14 and 19]). If we put a global texture onto such an oversimplified proxy, it will appear correct to a viewer *only* in the proximity of the input view that was used as texture for this scene part. In contrast, view-dependent texturing allows us (at rendering time) to choose input views for synthesis that are "compatible" with the virtual view due to being in its proximity, having roughly the same pixel footprint, etc.

An unstructured Lumigraph that uses per-view depth maps as proxy instead of a global polygon model does not only belong into the category of view-dependent texturing but also into that of view-dependent geometry. View-dependent geometry is advantageous in that the proxy has a geometric resolution compatible with the virtual view and it is disadvantageous in that depth maps typically contain more holes and outliers than a global polygon model.

A lesson learned from light fields and Lumigraphs is that proxies are essential for improving an algorithm's output quality without having to capture an immense

[29] It can be automated by using an array of cameras or (for much smaller camera translations) with specialized plenoptic cameras [Adelson and Wang 1992, Ng et al. 2005] that use micro-lenses to turn a regular camera sensor into an array of multiple cameras.

[30] Or if captured irregularly, they need to be "rebinned" into a grid which entails a quality loss.

number of pictures (and thereby creating an immense storage and computational burden). Chai et al. [2000] analyzed this in a principled way and found an inverse relationship between the amount of geometry (or rather its accuracy) and the number of images an IBR algorithm requires given that we want to keep the result quality and the movement liberties of the virtual camera constant. This is why Kang et al. [2000] also refer to the IBR continuum as *geometry-image continuum*.[31] If we want to model the proxy automatically, this geometry-vs-images statement must of course be taken down a notch because image-based modeling also requires lots of images.

Goals of IBR Systems: In the unstructured Lumigraph paper, Buehler et al. also formulate eight "goals" of IBR algorithms:

1. Use of geometric proxies: If available, proxies should be used for depth correction and visibility (*i.e.*, occlusion) reasoning. Input cameras placed very densely yield almost perfect virtual images even without a proxy, but since this is very expensive, we should alleviate errors with a proxy.
2. Epipole consistency: If a virtual ray goes through the center of an input camera and is within the field of view of that camera, that camera should be given a very high weight in the weighting scheme.
3. Resolution sensitivity: A virtual view should be synthesized from input views that have a similar pixel footprint. Otherwise, distant input views might be used to synthesize a close-up virtual view, producing a blurred result.
4. Unstructured input: Algorithms should be able to handle input cameras whose extrinsics have no underlying pattern, because structured input like the light field grid or the concentric mosaic circles are more complicated to acquire.
5. Equivalent ray consistency: A virtual ray should look identical no matter where along the ray the virtual camera is placed, unless this violates, *e.g.*, visibility constraints or resolution sensitivity.
6. Continuity: The algorithm should not introduce spatial (*i.e.*, within a virtual image) or temporal (*i.e.*, when moving the virtual camera) discontinuities.
7. Minimal angular deviation: When blending a virtual from input rays, input rays with a smaller angular difference to the virtual ray (as opposed to distance on the camera and focal plane as done in light fields) should get a higher weight.
8. Realtime: Rendering should be done at interactive frame rates.

View-dependent goals of course cannot be fulfilled by view-*in*dependent approaches. When, *e.g.*, constructing a global texture-mapped polygon model (as we will do in Chapter 3), it is impossible to produce a result that is consistent (see goal 2) with all input images' pixels unless the proxy is perfect and the scene's surface reflectance is purely Lambertian. We suppose that Buehler et al. did not have McMillan and Bishop's broad idea of image-based rendering in mind.

[31] Kang et al. [2006, Figure 2.2] point out that the Lumigraph [Gortler et al. 1996] "is a bit of an anomaly in this continuum, since it uses explicit geometry and a relatively dense set of images."

2.4. Markov Random Fields

Markov random fields are so versatile and frequently used in computer vision that we dare calling them the Swiss army knife of computer vision. Since we need some knowledge about them later in Chapter 4, this section gives a short introduction to them and sketches the ideas behind some basic algorithms to solve them.

2.4.1. Bayesian Inference

In computer vision we frequently deal with problems where some variables can be observed and others are hidden (*i.e.*, not directly observable) and "explain" the values of the observable variables. For example, in an optical flow problem the optical flow itself is hidden and its influence on the pixels' color values is observable. Or in an active depth camera, the actual depth is hidden and we can only observe the actual depth overlaid with noise. We treat hidden and observed variables as random variables and call the set of observed variables $\mathcal{O}$ and the set of hidden variables $\mathcal{H}$. Bayes' theorem gives us

$$P(\mathcal{H} \mid \mathcal{O})\, P(\mathcal{O}) = P(\mathcal{H}, \mathcal{O}) = P(\mathcal{O}, \mathcal{H}) = P(\mathcal{O} \mid \mathcal{H})\, P(\mathcal{H}) \tag{2.32}$$

$$\implies P(\mathcal{H} \mid \mathcal{O}) = \frac{P(\mathcal{O} \mid \mathcal{H})\, P(\mathcal{H})}{P(\mathcal{O})} \propto P(\mathcal{O} \mid \mathcal{H})\, P(\mathcal{H}). \tag{2.33}$$

The last step – dropping $P(\mathcal{O})$ – breaks the probability density function property, but does not affect the comparison of different values in $\mathcal{H}$. $\mathcal{O}$ is fixed to the observations we made and therefore $P(\mathcal{O})$ is only a normalization constant. The convenient consequence is that we do not need to come up with a model for $P(\mathcal{O})$. $P(\mathcal{H} \mid \mathcal{O})$ is called the posterior, $P(\mathcal{O} \mid \mathcal{H})$ is the likelihood, and $P(\mathcal{H})$ is the prior.

The likelihood is a model of how the observed values arise from the hidden values. Loosely speaking, we can say that it is a forward model. If we again consider the above-mentioned problem of recovering actual depth from depth measured with an active camera, we could, *e.g.*, model the measured depth as being Gaussian distributed around the actual depth with a certain standard deviation.[32]

The prior's purpose is to couple hidden variables together: If we consider the above depth problem again, most pixels tend to have a similar depth compared to their neighboring pixels. Large jumps in depth only occur at object boundaries and boundary pixels are much rarer than non-boundary pixels. Similarly, in optical flow estimation, most pixels have a similar flow compared to their neighbors unless they are at an object boundary. We call this piecewise smoothness. Problems such as image segmentation or foreground segmentation are even piecewise constant: Most pixels belong to the exact same segment as their neighbors. Many problems from computer vision and other fields have solutions that are piecewise smooth/constant instead of wildly oscillating. We already discussed the concept of regularization in

[32] In practice, one would probably not solve this problem with an MRF but rather with a filter, but for our purpose it makes for a relatively easy to understand example.

the context of multi-view stereo (Section 2.2.2.3). The prior in Bayesian inference is the stochastic equivalent of regularization.

We then seek the most likely posterior or maximum a posteriori (MAP) estimate

$$\begin{aligned} &\max_{\mathcal{H}} P(\mathcal{H} \mid \mathcal{O}) \propto \max_{\mathcal{H}} P(\mathcal{O} \mid \mathcal{H})\, P(\mathcal{H}) \\ \implies &\arg\max_{\mathcal{H}} P(\mathcal{H} \mid \mathcal{O}) = \arg\max_{\mathcal{H}} P(\mathcal{O} \mid \mathcal{H})\, P(\mathcal{H}). \end{aligned} \tag{2.34}$$

The power of Equations (2.33) and (2.34) is that even if we only have a forward model of how observed arise from hidden values, we can still invert this and infer hidden from observed values. Because most computer vision problems are inverse problems in this sense, MRFs are being used in many computer vision fields such as (multi-view) stereo, global structure from motion [Crandall et al. 2013], segmentation, and many many more.

2.4.2. Pairwise Markov Random Fields

In general, the explicit forms of $P(\mathcal{O} \mid \mathcal{H})$ and $P(\mathcal{H})$ can be very complicated. To end up with formulations that are simple enough to be modeled (*e.g.*, where one can experimentally collect enough ground-truth data to back up that a certain model is a reasonable approximation to the real world) and to be solved (*i.e.*, MAP estimation can be done without falling into poor local optima), a simplifying assumption is frequently used in computer vision and other fields: It is assumed that the random variables and their dependencies can be modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where each random variable is represented by exactly one vertex of the vertex set $\mathcal{V}$ and each vertex (*i.e.*, random variable) is conditionally independent of all other vertices given its immediate neighbors – the so-called local Markov property ($\{a, b\}$ is used to denote unordered pairs):

$$P(v \mid \mathcal{V} \setminus v) = P(v \mid \{n : \{v, n\} \in \mathcal{E}\}). \tag{2.35}$$

The neighbors $\{n : \{v, n\} \in \mathcal{E}\}$, v's so-called Markov blanket, "shield" v from the influence of more distant vertices. The underlying idea is to "explicitly represent only the associations between relatively few pairs of [variables] – those [variables] that are defined as neighbors because of sharing an edge in $\mathcal{E}$. The great attraction of Markov models is that they leverage a knock-on effect – that explicit short-range linkages give rise to implied long-range correlations. Thus correlations over long ranges [...] can be obtained without undue computational cost" [Blake et al. 2011].

In the following, $\underline{\times}$ is an operator that is similar to the Cartesian product but produces unordered instead of ordered pairs: $A \underline{\times} B = \{\{a, b\} : a \in A, b \in B\}$. If the Markov property is fulfilled, prior and likelihood can be written as a product of bivariate probability density functions $P_S(h_i, h_j)$ and univariate probability density

functions $P_{D_{o_i}}(h_j)$,[33] respectively:

$$P(\mathcal{H}) = \prod_{\{h_i,h_j\} \in \mathcal{E} \cap (\mathcal{H} \underline{\times} \mathcal{H})} P_S(h_i, h_j) \tag{2.36}$$

$$P(\mathcal{O} \mid \mathcal{H}) = \prod_{\{o_i,h_j\} \in \mathcal{E} \cap (\mathcal{O} \underline{\times} \mathcal{H})} P_{D_{o_i}}(h_j). \tag{2.37}$$

With this the MAP estimate (Equation (2.34)) becomes

$$\begin{aligned} \arg\max_{\mathcal{H}} P(\mathcal{H} \,|\, \mathcal{O}) &= \arg\max_{\mathcal{H}} P(\mathcal{O} \mid \mathcal{H})\, P(\mathcal{H}) \\ &= \arg\max_{\mathcal{H}} \prod_{\{o_i,h_j\} \in \mathcal{E} \cap (\mathcal{O} \underline{\times} \mathcal{H})} P_{D_{o_i}}(h_j) \cdot \prod_{\{h_i,h_j\} \in \mathcal{E} \cap (\mathcal{H} \underline{\times} \mathcal{H})} P_S(h_i, h_j). \end{aligned} \tag{2.38}$$

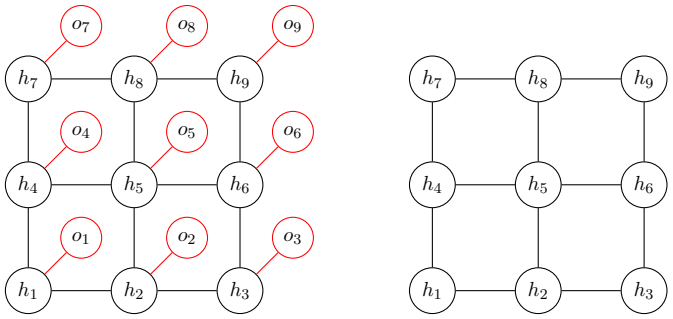


Figure 2.9.: *Left:* A simple MRF with hidden (black) and observed (red) variables. Each observed is paired up with one hidden variable. This particular MRF has a regular, 2-dimensional, 4-connected grid structure which is typical for computer vision problems that are based on images. In general, however, MRFs can have an arbitrary, even irregular structure. *Right:* The same MRF but without the observed variables. This reduced form is the one that is usually used when talking, *e.g.*, about the regularity or irregularity of an MRF's topology.

Looking at hidden and observed variables again, most models pair up one observed with one hidden variable (as shown on the left of Figure 2.9), *i.e.*, $|\mathcal{O}| = |\mathcal{H}|$ and $\mathcal{E} = \mathcal{E}_{\mathcal{O},\mathcal{H}} \cup \mathcal{E}_{\mathcal{H},\mathcal{H}}$, where $\mathcal{E}_{\mathcal{O},\mathcal{H}}$ is an (unordered) bijection between $\mathcal{O}$ and $\mathcal{H}$ and $\mathcal{E}_{\mathcal{H},\mathcal{H}} \subseteq \mathcal{H} \underline{\times} \mathcal{H}$. Since observed variables are fixed and neatly paired with hidden variables, one often simplifies the MRF by omitting $\mathcal{O}$ from $\mathcal{V}$ and $\mathcal{E}_{\mathcal{O},\mathcal{H}}$ from $\mathcal{E}$ (shown on the right of Figure 2.9) and accounting for them implicitly (by encoding them in the $P_{D_{o_i}}(h_j)$). Then, the MAP estimate is somewhat easier on the eyes:

$$\arg\max_{\mathcal{H}} P(\mathcal{H} \mid \mathcal{O}) = \arg\max_{\mathcal{H}} \prod_{h_i \in \mathcal{H}} P_{D_{o_i}}(h_i) \cdot \prod_{\{h_i,h_j\} \in \mathcal{E}} P_S(h_i, h_j). \tag{2.39}$$

[33] One may think of $P_{D_{o_i}}(h_j)$ as being bivariate, too, but since the observed variables are fixed, their values are encoded as constants in the functions. Hence the $_{o_i}$ subscript.

Since there is now a one-to-one mapping between hidden variables and vertices, in the following we switch from probability to graph terminology: We (slightly imprecisely) say that $v_i = h_i$, rename the value of a variable h_i into "label" ℓ_{v_i} of vertex v_i (and the value of o_i into ℓ_{o_i}), call the set of possible labels the label set $\mathcal{L}$ (in computer vision it is typically discrete and finite: $\mathcal{L} = \{0, \dots, \ell_{\max}\}$), rename $P_{D_{o_i}}(h_i)$ as $D_{v_i}(\ell_{v_i})$, and $P_S(h_i, h_j)$ as $S_{v_i,v_j}(\ell_{v_i}, \ell_{v_j})$. This yields

$$\arg\max_{\mathcal{H}} P(\mathcal{H} \mid \mathcal{O}) = \arg\max_{(\ell_{v_1},\dots,\ell_{v_n}) \in \mathcal{L}^n} \prod_{v_i \in \mathcal{V}} D_{v_i}(\ell_{v_i}) \cdot \prod_{\{v_i,v_j\} \in \mathcal{E}} S_{v_i,v_j}(\ell_{v_i}, \ell_{v_j}). \quad (2.40)$$

D and S are often Gaussian-like functions

$$D_{v_i}(\ell_{v_i}) \propto e^{-(2\sigma_d^2)^{-1} d(\ell_{o_i} - \ell_{v_i})} \text{ and} \quad (2.41)$$

$$S_{v_i,v_j}(\ell_{v_i}, \ell_{v_j}) \propto e^{-(2\sigma_s^2)^{-1} s(\ell_{v_i} - \ell_{v_j})}, \quad (2.42)$$

with distance functions (*e.g.*, metrics) d and s[34]:

$$\arg\max_{\mathcal{H}} P(\mathcal{H} \mid \mathcal{O}) = \arg\max_{(\ell_{v_1},\dots,\ell_{v_n}) \in \mathcal{L}^n} \prod_{v_i \in \mathcal{V}} e^{-(2\sigma_d^2)^{-1} d(\ell_{o_i} - \ell_{v_i})} \cdot \prod_{\{v_i,v_j\} \in \mathcal{E}} e^{-(2\sigma_s^2)^{-1} s(\ell_{v_i} - \ell_{v_j})}. \quad (2.43)$$

For numerical stability, this is usually solved in (negative) logarithmic space:

$$\arg\max_{\mathcal{H}} P(\mathcal{H} \mid \mathcal{O}) = \arg\min_{\mathcal{H}} \left(-\log P(\mathcal{H} \mid \mathcal{O}) \right) \quad (2.44)$$

$$= \arg\min_{(\ell_{v_1},\dots,\ell_{v_n}) \in \mathcal{L}^n} \sum_{v_i \in \mathcal{V}} \frac{d(\ell_{o_i} - \ell_{v_i})}{2\sigma_d^2} + \sum_{\{v_i,v_j\} \in \mathcal{E}} \frac{s(\ell_{v_i} - \ell_{v_j})}{2\sigma_s^2} \quad (2.45)$$

$$= \arg\min_{(\ell_{v_1},\dots,\ell_{v_n}) \in \mathcal{L}^n} \sum_{v_i \in \mathcal{V}} d(\ell_{o_i} - \ell_{v_i}) + \frac{2\sigma_d^2}{2\sigma_s^2} \sum_{\{v_i,v_j\} \in \mathcal{E}} s(\ell_{v_i} - \ell_{v_j}) \quad (2.46)$$

$$= \arg\min_{(\ell_{v_1},\dots,\ell_{v_n}) \in \mathcal{L}^n} \sum_{v_i \in \mathcal{V}} d(\ell_{o_i} - \ell_{v_i}) + \lambda \sum_{\{v_i,v_j\} \in \mathcal{E}} s(\ell_{v_i} - \ell_{v_j}). \quad (2.47)$$

Thus, the maximization is turned into a minimization problem. This is the canonical form of (pairwise) Markov random fields. The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is called the MRF's *topology*. A vector $(\ell_{v_1}, \dots, \ell_{v_n}) \in \mathcal{L}^n$ is called a *solution* of the MRF, everything behind the arg min is called its *energy*, and the solution with minimal energy is the *optimal solution*. The function $d(\cdot)$ corresponds to the likelihood and is called the *data term* (or data cost) because it forces the hidden variables to obey the observed data. The function $s(\cdot)$ corresponds to the prior and is called the *smoothness term* (or smoothness cost) because it enforces smoothness of the solution as discussed in Section 2.4.1. The coefficient λ can be used to control the degree of solution

[34] In general, d and s can be spatially varying, *i.e.*, d_{v_i} and s_{v_i,v_j}, but for the sake of notational simplicity we assume they are identical for all vertices and edges.

smoothness. An algorithm that solves the MAP problem on Equation (2.47) is called an MRF solver. Many solvers have requirements on the properties of the smoothness term. *E.g.*, α-expansion, which we explain in Section 2.4.3, requires s to be submodular. This rules out for example anti-Potts, anti-metric, or discrete, learned cost functions.

Typical choices for d and s include the L_2, L_1, or Huber norm, or the Potts model. The latter three are said to be "more robust" than the L_2 norm because they do not punish large distances excessively. In the data term this can be important because, depending on the observation noise model, a large difference between hidden and observed variable may not be much less likely than a small (but non-zero) difference. Similarly, in the smoothness term large jumps in the solution may not be much less likely than small (but non-zero) jumps.[35]

In addition to data and smoothness costs one may include so-called label costs:

$$\underset{(\ell_{v_1},\dots,\ell_{v_n})\in\mathcal{L}^n}{\arg\min} \sum_{v_i\in\mathcal{V}} d(\ell_{o_i}-\ell_{v_i}) + \lambda \sum_{\{v_i,v_j\}\in\mathcal{E}} s(\ell_{v_i}-\ell_{v_j}) + \sum_{\ell\in\mathcal{L}} l(\ell) \tag{2.48}$$

$$l(\ell) = \begin{cases} c_\ell & \text{if } \exists\, \ell_{v_i} : \ell_{v_i} = \ell \\ 0 & \text{else} \end{cases} \tag{2.49}$$

where l is a function that introduces a constant cost c_ℓ for each label that is used in the solution, which effectively promotes label sparsity in the solution. Shan et al. [2014b], for example, use it to keep the maximum number of input photos for panorama stitching low in order to have as few stitching seams as possible. However, many solvers do not support label costs.

Since the MAP estimation problem is in general $\mathcal{NP}$-hard [Shimony 1994], large, realistic problems need to be solved approximately. Many MRF solvers start with a suboptimal solution and use a heuristic to iteratively improve it until they reach a local optimum. Some of these heuristics have guarantees on the quality of their solution, *e.g.*, α-expansion guarantees the local optimum to be within a constant factor of the global optimum (if the smoothness term is submodular). Two important classes of iterative algorithms are algorithms that either reduce the MRF's label space (Section 2.4.3) or its topology (Section 2.4.4) in each iteration.

2.4.3. Solvers with Label Space Reduction (α-Expansion)

α-expansion [Boykov et al. 2001] is a popular and good, yet easy to understand solver, which is why we chose to explain it here. Most MRFs have a label set larger than two labels, but let us assume for a minute that they only have two possible labels 0 and 1. Such two-label MRFs can be solved with the graph cut algorithm which we will treat as a black box here. It suffices to say that, given an MRF and

[35] In object segmentation, for example, labels correspond to real-world objects (*e.g.*, a tree, the street, a car). Jumping from the street to a tree or a car may be equally likely. How objects are encoded in labels is arbitrary, therefore the "distance" between labels is meaningless and a (non-zero) jump should always give the same smoothness costs.

a vector $(\{0,1\},\ldots,\{0,1\})$ of admissible labels per vertex as input, it computes the optimal solution $(l_1,\ldots,l_n)$ in polynomial time (in the number of edges):

Listing 2.1: Pseudocode for graph cut

```
1 (l_1,...,l_n) ← graph_cut(MRF,({0,1},...,{0,1}))
```

Now, what α-expansion [Boykov et al. 2001] does is the following:

Listing 2.2: Pseudocode for α-expansion

```
1 (l_1,...,l_n) ← (0,...,0)
2 repeat
3    foreach (label α ∈ {0,...,max})
4       (l_1,...,l_n) ← graph_cut(MRF,({l_1,α},...,{l_n,α}))
5 until (the last iteration did not decrease the energy)
```

It first initializes $(l_1,\ldots,l_n)$ with some solution (instead of $(0,\ldots,0)$ it could also be a random initialization). The algorithm then enters an outer loop of optimization rounds and an inner loop which tries out all possible labels.[36] In the inner loop, graph cut lets each vertex "choose" to switch to the new label α instead of its old label. Thereby, α-expansion reduces the full optimization to a series of optimizations on a label space of size two.

α-expansion repeats the outer loop until an entire round is completed with no energy improvement. The final energy is not guaranteed to be the global optimum, but it is within a factor of $2c$ of the optimum, where c is the ratio between the largest and the smallest non-zero value that the smoothness term can have. In practice $2c$ is, however, not a very tight bound.

Lempitsky et al. [2010] developed a replacement for graph cut that can handle non-submodular smoothness terms and Delong et al. [2012] incorporated label costs into α-expansion.

2.4.4. Solvers with Topology Reduction (Block Coordinate Descent)

An alternative to reducing the label space in each solver iteration is to reduce the MRF's topology: One selects a subset of all vertices, keeps the remainder fixed (*i.e.*, retains their labels from the previous solver iteration), and only optimizes on the selected set while still taking the fixed vertices and their labels into account (*i.e.*, adding their data and smoothness costs to the energy[37]). This is known as block coordinate descent (BCD) and the selected vertices are called coordinates. After one BCD iteration, a new set of coordinates is selected, optimized, and this is repeated (with a new set of coordinates in each iteration) until some stopping criterion (elapsed time, iterations, no energy decrease, *etc.*) is fulfilled.

[36] For some problems it is helpful to make the inner loop perform randomly permuted, rather than fixed, iterations over the labels.

[37] It is sufficient to add the smoothness costs between fixed and selected vertices. Data costs of fixed vertices and smoothness costs between two fixed vertices need not be added because they are fixed and play no role in the optimization.

The crucial problem in BCD is to select coordinates such that optimizing on the reduced topology a) remains tractable and b) allows for large energy reductions. Selecting, *e.g.*, all vertices as coordinates yields the original problem and allows for maximum energy decrease but would be in general $\mathcal{NP}$-hard again. The simplest BCD representative is iterated conditional modes (ICM) [Besag 1986] which selects just one vertex in each iteration. Besag [1986, Section 2.6 "Some Modifications of ICM"] already remarked that this may be generalized: "A more radical suggestion is to replace single [vertex] optimization [...] by maximization over a (small) set of contiguous [vertices ...]. We have in mind, perhaps, blocks B of four or five [vertices], though even here exhaustive search can be time consuming". 20 years later, armed with the compute power increases from 13 iterations of Moore's law, Kelm et al. [2006] put this into action. The problem with choosing densely connected blocks (like Besag's or Kelm et al.'s rectangular blocks in a grid MRF) is that this does not reduce the computational complexity; it only decreases the problem's size.

One alternative that *does* reduce the complexity are scanlines: Chen and Koltun [2014] select one vertical or horizontal pixel line of an image as coordinate set and solve that line with dynamic programming. The complexity of this is $O(|\mathcal{C}||\mathcal{L}|^2)$ per solver iteration ($|\mathcal{C}|$ being the size of the coordinate set) or even $O(|\mathcal{C}||\mathcal{L}|)$ [Felzenszwalb and Huttenlocher 2006] for certain smoothness cost types such as L_1 norm, L_2 norm, Huber norm, or the Potts model. Moreover, they can solve multiple scanlines in parallel provided that the lines solved in parallel do not interact with each other: First, all uneven image rows are solved in parallel (with one CPU core optimizing one row), then all even rows in parallel, then all uneven columns, then all even columns, and all over again until the stopping criterion is fulfilled. As long as two coordinate lines are separated by a line of fixed variables, they cannot interact. If two coordinate lines were to touch, we would again obtain a loopy coordinate graph, for which Besag had no better method than solving it exhaustively.

Veksler [2005] takes this idea further – from lines to trees: She samples a spanning tree from the MRF and solves it with dynamic programming in $O(|\mathcal{C}||\mathcal{L}|^2)$ [Veksler 2005],[Szeliski 2010, Sec. B.5.2] (or $O(|\mathcal{C}||\mathcal{L}|)$ for certain smoothness cost types).

Part I.

Creating Realistic Textured 3D Reconstructions

3. Texturing Polygonal 3D Models from Registered Images

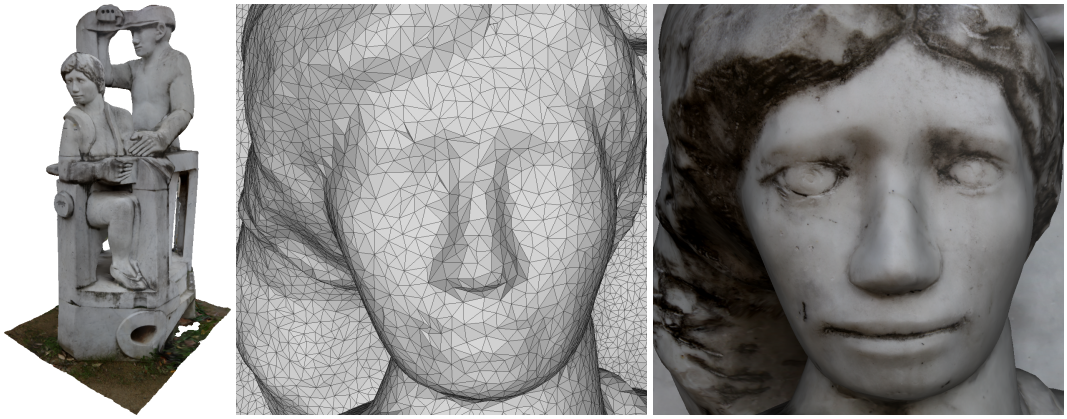


Figure 3.1.: *Left to right:* Automatically textured model reconstructed from a set of images, mesh close-up, and the same mesh rendered with texture.

In the last decade, 3D reconstruction from images has made tremendous progress. Camera calibration is now possible on Internet photo collections [Snavely et al. 2006], even on city-scale [Agarwal et al. 2009] and world-scale datasets [Heinly et al. 2015]. There is a wealth of dense multi-view stereo reconstruction algorithms, some also scaling to city level [Frahm et al. 2010, Furukawa et al. 2010] or world level [Schönberger et al. 2016]. Realism is strongly increasing: Shan et al. [2013] presented large reconstructions which are hard to distinguish from the input images if rendered at a low resolution. Looking at the impressive results of state of the art reconstruction algorithms mentioned above, one notices, however, that color information is still encoded as per-vertex color and therefore coupled to the mesh resolution. An important building block to make the reconstructed models a convincing experience for end users while keeping their size manageable is still missing: texture. Although textured models are common in the context of computer graphics, texturing 3D reconstructions from images is very challenging due to illumination and exposure changes between images, non-rigid scene parts, unreconstructed occluding objects, and image scales varying by orders of magnitude between images.[38]

So far, texture acquisition has not attracted nearly as much attention as geom-

[38] Consider, *e.g.*, the difference between a distant overview shot where one pixel covers many centimeters in the scene and a close-up photo where a pixel covers less than a millimeter.

etry acquisition: Current benchmarks such as the Middlebury multi-view stereo benchmark [Seitz et al. 2006] focus only on geometry and ignore appearance aspects. Furukawa et al. [2010] produce and render point clouds with very limited resolution, which is especially apparent in close-ups. To texture their reconstructed geometry, Frahm et al. [2010] use the mean of all images that observe it which yields insufficient visual fidelity. Shan et al. [2013] perform impressive work on estimating lighting parameters per input image and per-vertex reflectance parameters. Still, they use per-vertex colors and are therefore limited to the mesh resolution. Our texturing abilities seem to be lagging behind those of geometry reconstruction.

While there exists a significant body of work on texturing (Section 3.1 gives a review), most texture creation methods focus on small, controlled datasets and are unable to handle the complexity and size of large-scale 3D reconstructions. Prominent exceptions handle only specialized cases such as architectural scenes: Garcia-Dorado et al. [2013] reconstruct and texture entire cities, but their method is specialized to the city setting as it uses a 2.5D scene representation (building outlines and estimated elevation maps) and a sparse image set where each mesh face is visible in very few views. Furthermore, they are restricted to regular block city structures with planar surfaces and treat buildings, ground, and building-ground transitions differently during texturing. Sinha et al. [2008] texture large 3D models with planar surfaces (*i.e.*, buildings) that were created interactively using cues from structure from motion. Since they only consider this planar case, they can optimize each surface independently. In addition, they rely on user interaction to mark occluding objects (*e.g.*, trees) in order to ignore them during texturing. Similarly, Tan et al. [2008] propose an interactive texture mapping approach for building façades.

We argue that texture reconstruction is vitally important for creating realistic models without increasing their geometric complexity. It should ideally be fully automatic even for large-scale, real-world datasets. This is challenging due to the properties of the input images as well as unavoidable imperfections in the reconstructed geometry. Additionally, a practical method should be efficient enough to handle even large models in a reasonable time frame. We therefore present the first comprehensive texturing framework for large-scale, image-based 3D models. It requires a 3D polygon mesh and photos that are registered (*i.e.*, their internal and external camera parameters are known) in the same coordinate frame as the mesh and it produces a global, texture-mapped 3D mesh.[39] The input mesh can either be obtained through a classical image-based modeling pipeline as described in Section 2.2, or through actively scanning a scene, taking photos of the same scene, and registering the photos against the scan.[40] Our method fully automat-

[39] It therefore belongs far into the geometry end of Kang et al.'s IBR continuum, see page 30.

[40] The first path, image-based modeling, is elegant because the input images' parameters and the mesh are automatically in the same coordinate frame. However, many applications follow the second path [Krispel et al. 2015, Devaux and Brédif 2016, Heindl et al. 2016a,b, Labrie-Larrivée et al. 2016], especially in cases where image-based modeling performs poorly. The mesh is obtained with an active sensor (structured light, LiDAR, *etc.*) and surface reconstruction [Newcombe et al. 2011] and the color images are taken with a camera with fixed and known offset to

ically accounts for typical challenges inherent in this setting: the large number of input images, their drastically varying properties such as image scale, (out-of-focus) blur, exposure variation, and occluders (*e.g.*, moving plants or pedestrians). Using the proposed technique, we are able to texture datasets that are several orders of magnitude larger and far more challenging than those shown in prior work.

After reviewing related work in Section 3.1, we will in Section 3.2 explain the algorithm by Lempitsky and Ivanov [2007] on which ours is based, before describing our algorithm in Section 3.3. In Section 3.4 we evaluate our algorithm's runtime as well as the implications of algorithmic decisions on the textured result.

3.1. Related Work

Texturing a 3D model from multiple registered images is typically performed in a two step approach: First, one needs to select which view(s) should be used to texture each face, yielding a preliminary texture. In the second step, this texture is optimized for consistency to avoid seams between adjacent texture patches.

3.1.1. View Selection

Regarding which view is selected to be used as texture for each part of the model's surface, the literature can be divided into two main classes: Several approaches select and blend multiple views per mesh face to achieve a consistent texture across patch borders [Grammatikopoulos et al. 2007, Callieri et al. 2008, Pagés et al. 2015, Maier et al. 2015, Ley and Hellwich 2016, Liu et al. 2016]. In contrast, many others texture each mesh face with exactly one view [Lempitsky and Ivanov 2007, Velho and Sossai Jr. 2007, Gal et al. 2010, Garcia-Dorado et al. 2013, Heindl et al. 2016b]. Sinha et al. [2008] also select one view, but per texel instead of per face. Some authors [Allène et al. 2008, Chen et al. 2012] propose hybrid approaches that generally select a single view but blend multiple views in the vicinity of texture patch borders.

Blending images causes problems in a multi-view stereo setting: First, if camera parameters or the reconstructed geometry are slightly inaccurate, texture patches may be misaligned at their borders, produce ghosting, and result in strongly visible seams. This also occurs if the geometric model has a relatively low resolution and does not perfectly represent the true object geometry. Second, in realistic multi-view stereo datasets we often observe a strong difference in image scale: The same face may cover less than one pixel in one view and several thousand in another. If these views are blended, distant views blur out details from close-ups. This can be alleviated by weighting the images to be blended [Callieri et al. 2008] or by blending in frequency space [Allène et al. 2008, Chen et al. 2012].

the active sensor. The color images can even be taken independently and subsequently be registered [Plötz and Roth 2017]. Judging from feedback from various researchers after publication of our texturing work in 2014b, texturing *scanned* models seems to be a very frequent use case.

Callieri et al. [2008] compute weights for blending as a product of masks indicating the suitability of input image pixels for texturing with respect to viewing angle, proximity to the surface, and proximity to depth discontinuities.[41] Similarly, Grammatikopoulos et al. [2007] and Liu et al. [2016] blend based on viewing angle and proximity to the surface, and Pagés et al. [2015] blend based on triangle projection area which indirectly incorporates viewing angle and proximity. Wang et al. [2001] derive theoretically optimal blending weights using image processing theory and obtain weights that decrease linearly with an increasing viewing angle.

Methods that are directly related in that they also compute blending weights according to the geometric configuration of views and surface, are the unstructured Lumigraph [Buehler et al. 2001] and its successors. They are, however, IBR algorithms with view-dependent texture – see Section 2.3. While these can give the illusion of the scene having more geometric detail than the underlying proxy actually has, they require specialized rendering algorithms and therefore cannot always be easily integrated into traditional rendering pipelines such as those in video games. Thus, in games and many other applications, static geometry with static texture is still the scene representation of choice.

The texturing algorithm of Lempitsky and Ivanov [2007] selects a single view per mesh face based on a pairwise Markov random field. Its data term judges the quality of views for texturing while their smoothness term models the severity of seams between texture patches. Velho and Sossai Jr. [2007], Allène et al. [2008] and Gal et al. [2010] proposed data terms that incorporate additional effects compared to Lempitsky and Ivanov's data term. Since these methods form the basis of our technique, we give a more detailed description of them in Section 3.2. Wang and Geng [2017] presented a technique that is largely similar to Lempitsky and Ivanov's but incorporates the virtual view position into the view selection, which makes it a (very compute-intensive) view-dependent texturing method.

A convenient property of all methods that select one view per mesh face is that the mesh parameterization – *i.e.*, the way in which a mesh is cut into patches/charts/pieces and in which each point on a patch is mapped into a 2D texture atlas – follows directly from the view selection: Each set of contiguous mesh faces textured from the same view forms one texture patch. Such a patch is then projected into its respective input image, a vertex's projection's pixel coordinates are used as its texture coordinates[42], and the content of the texture atlas is a direct copy from the input image. In the case that multiple views are blended to form the texture of a face, the parameterization is a bit more involved: The mesh needs to be cut open subject to certain optimization criteria, *e.g.*, minimizing texture stretch while also minimizing seam length (see Sheffer et al. [2006] for a survey). Afterwards, the input images need to be resampled to their new coordinate frame (*i.e.*, the texture atlas)

[41] Callieri et al. [2008] do not, however, compute real textures, but suggest increasing the mesh's geometric resolution through mesh subdivision and then using vertex colors. This contradicts the purpose of textures – high visual resolution at a low geometric resolution for a low memory footprint of the whole model – and is not feasible for large datasets or high-resolution images.

[42] *I.e.*, the mapping from a mesh vertex to texture atlas coordinates.

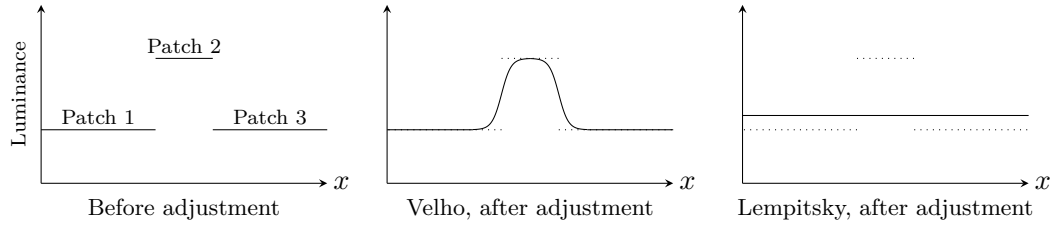


Figure 3.2.: Color adjustment (here in 1D). *Left:* Patch 2 is lighter than Patch 1 and 3, *e.g.*, due to different exposure. *Center:* Velho and Sossai Jr. [2007] let the luminance transition smoothly but in essence each patch gets to keep its luminance. *Right:* Lempitsky and Ivanov [2007] compute globally optimal adjustments.

and blended using their blending weights.

3.1.2. Color Adjustment

After view selection, the resulting texture patches may have strong color discontinuities due to, *e.g.*, exposure and illumination differences. Thus, adjacent texture patches need to be photometrically adjusted so that their seams become less noticeable.

This can be done either locally or globally. Velho and Sossai Jr. [2007] (Figure 3.2 (center)) adjust locally by setting the color at a seam to the mean of the left and right patch. They then use heat diffusion to achieve a smooth color transition towards this mean, which noticeably lightens Patches 1 and 3 at their borders. In contrast, Lempitsky and Ivanov [2007] compute globally optimal luminance correction terms that are added to the vertex luminances subject to two intuitive constraints: After adjustment, luminance differences at seams and the derivative of adjustments within a texture patch should both be small. This allows for a correction where Patch 2 is adjusted to the same level as Patch 1 and 3 (Figure 3.2 (right)) without visible meso- or large-scale luminance changes.

3.2. Assumptions and Base Method

Our method takes as input a set of (typically several hundred) images of a scene that were registered using structure from motion [Snavely et al. 2006]. Based on this, the scene geometry is reconstructed using multi-view stereo (*e.g.*, Frahm et al. [2010], Furukawa et al. [2010]) and surface reconstruction (*e.g.*, [Kazhdan et al. 2006, Fuhrmann and Goesele 2014]), yielding a good (but not necessarily perfect) quality triangular mesh. This setting ensures that the images are registered against the 3D reconstruction, but also yields some inherent challenges: The structure from motion camera parameters may not be perfectly accurate and the reconstructed geometry may not represent the underlying scene perfectly. Furthermore, the input images may exhibit strong illumination, exposure, and scale differences and contain unreconstructed occluders such as pedestrians.

We now give an overview over how Lempitsky and Ivanov [2007] and some related algorithms work since our approach is based on their work. Section 3.3 describes the key changes made in our approach to handle the above challenges.

The initial step in the pipeline is to determine the visibility of faces in the input images. Lempitsky and Ivanov then compute a labeling $(\ell_1, \ldots, \ell_n)$ that assigns a view ℓ_i to be used as texture for each mesh face F_i using a pairwise Markov random field formulation (we use a simpler notation than Lempitsky and Ivanov here):

$$E(\ell_1, \ldots, \ell_n) = \sum_{F_i \in \text{Faces}} E_{\text{data}}(F_i, \ell_i) + \sum_{(F_i, F_j) \in \text{Edges}} E_{\text{smooth}}(F_i, F_j, \ell_i, \ell_j). \quad (3.1)$$

Data Term: The data term E_{data} prefers "good" views for texturing a face. The smoothness term E_{smooth} minimizes the visibility of seams (*i.e.*, edges between faces textured with different images).

For the data term, the base method as well as Velho and Sossai Jr. [2007] use the angle between the viewing direction and the face normal. This is, however, insufficient for our datasets as it chooses images regardless of their proximity to the object, their resolution, or their out-of-focus blur. Allène et al. [2008] project a face into a view and use the projection's size as data term. This accounts for view proximity, viewing angle (because a face has a larger projection area in orthogonal views), and image resolution. The unstructured Lumigraph's [Buehler et al. 2001] view blending weights and the weights of texturing methods that blend multiple views per triangle (*e.g.*, Pagés et al. [2015]) are very much related to this because they account for the very same effects. However, the base method, Allène et al., and the unstructured Lumigraph do not consider the actual image content and therefore do not account for out-of-focus blur: In a close-up, the faces closest to the camera have a large projection area and are preferred by Allène's data term or the unstructured Lumigraph weights, but they may not be in focus and may lead to a blurry texture. Thus, Gal et al. [2010] use the gradient magnitude of the image integrated over the face's projection. This term is large if the projection area is large (close, orthogonal images with a high resolution) or the gradient magnitude is large (in-focus images).

Gal et al. also introduce two additional degrees of freedom into the data term: They improve the alignment of neighboring patches to minimize seam visibility by allowing input images to be translated by up to 64 pixels in x- or y-direction via additional MRF labels in a coarse to fine refinement.[43] We abstain from this because it only minimizes seam visibility and does not necessarily perform better in explaining the input data.[44] In a rendering of such a model, a texture patch would have an offset compared to its source image. Furthermore, these additional degrees

[43] An alternative is optical flow [Dellepiane et al. 2012, Boukhayma et al. 2016], which may in turn use MRFs in a coarse to fine manner as well.

[44] Later, in Part II of this thesis, we require that models explain the input images, or more precisely, that they explain hidden test images.

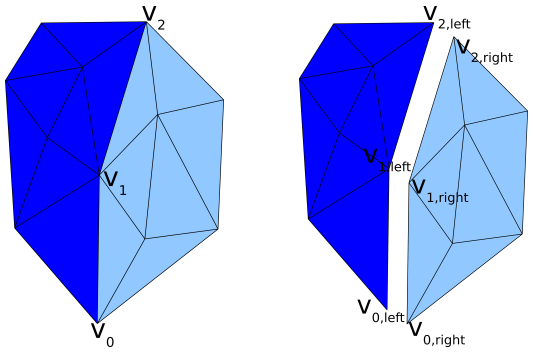


Figure 3.3.: For luminance adjustment, *i.e.*, making the dark blue match the light blue patch, we (virtually) split each of the seam vertices v_0, v_1, and v_2 into two vertices.

of freedom increase the computational costs such that the optimization may become infeasible for very large datasets.

Smoothness Term: Lempitsky and Ivanov's smoothness term is the difference between the texture to a seam's left and right side integrated over the seam. This should favor seams in regions where cameras are accurately registered or where misalignments are unnoticeable because the texture is smooth. We found that computation of the seam error integrals is a computational bottleneck and cannot be precomputed due to the prohibitively large number of combinations. Furthermore, it favors distant or low-resolution views since a blurry texture produces smaller seam errors, an issue that does not occur in Lempitsky and Ivanov's controlled lab datasets.

Optimization: The base method optimizes Equation (3.1) with GCO (graph cut optimization with α-expansion) [Boykov et al. 2001]. This can of course be done with any Markov random field solver which handles arbitrary MRF topologies. In Section 4 we present a faster alternative to GCO.

Color Adjustment: After obtaining a labeling from minimizing Equation (3.1), the patch colors are adjusted as follows: First, it must be ensured that each mesh vertex belongs to exactly one texture patch. This is illustrated in Figure 3.3: All vertices on seams – v_0, v_1, and v_2 – are duplicated into two vertices: $v_{0/1/2,\,\text{left}}$ to the left and $v_{0/1/2,\,\text{right}}$ to the right of the seam.[45] It is sufficient to make this duplication virtually, *i.e.*, to not actually create two vertices but to create two optimization variables for each seam vertex. After duplication, each vertex v has a unique color f_v and we search an additive adjustment g_v for v. Thus, v's adjusted color is

[45] In the following, we only consider the case where seam vertices belong to $n = 2$ patches. For $n > 2$ we create n copies of the vertex and optimize all pairs of those copies jointly, yielding a correction factor per vertex and patch.

$f_v + g_v$.[46] These adjustments need to minimize the following expression (we use a simpler notation than Lempitsky and Ivanov for clarity):

$$\min_{\boldsymbol{g}} \sum_{\substack{\text{all vertices } v_i \text{ lying} \\ \text{on a seam (where } v_i \\ \text{has been split into} \\ v_{i,\text{left}} \text{ and } v_{i,\text{right}})}} \Big((f_{v_{i,\text{left}}} + g_{v_{i,\text{left}}}) - (f_{v_{i,\text{right}}} + g_{v_{i,\text{right}}}) \Big)^2 + \lambda^2 \sum_{\substack{\text{all node pairs} \\ \{v_i, v_j\} \text{ that are} \\ \text{adjacent and in} \\ \text{the same patch}}} \Big(g_{v_i} - g_{v_j} \Big)^2. \tag{3.2}$$

The first term ensures that the adjusted color to a seam's left ($f_{v_{i,\text{left}}} + g_{v_{i,\text{left}}}$) and its right ($f_{v_{i,\text{right}}} + g_{v_{i,\text{right}}}$) are as similar as possible. The second term minimizes adjustment differences between adjacent vertices in the same texture patch. This favors adjustments that change as smoothly as possible within a texture patch.[47] After finding optimal g_v for all vertices, the corrections for each texel are interpolated from the g_v of its surrounding vertices using barycentric coordinates. Finally, the corrections are added to the input images, the texture patches are packed into texture atlases, and texture coordinates are attached to the vertices.

3.3. Large-Scale Texturing Approach

Following the base method we now explain our approach, focusing on the key novel aspects that we introduce to handle the challenges of real-world 3D reconstructions.

3.3.1. Preprocessing

For each combination of faces and views we determine whether the view sees the face by first discarding a large number of faces through back face and view frustum culling. Afterwards, we check for occlusions by shooting rays from the camera center to all three corners of the face in question and checking whether the ray intersects any other part of the input geometry on its way. This is more accurate than using rendering as done, *e.g.*, by Velho and Sossai Jr. [2007] and Callieri et al. [2008], and it has no relevant negative impact on performance, since it is not a bottleneck of our texturing algorithm. For all remaining face-view combinations we then precompute the data terms for Equation (3.1) since they are used multiple

[46] We note that additive per-vertex adjustments are not physically justifiable. Assuming a constant camera and constant illumination, a multiplicative per-image adjustment would be correct. If we wanted a more flexible model that allows for more effects, we could, *e.g.*, choose a model with a multiplicative and an additive per-image adjustment [Shen et al. 2016]. However, we will stick to Lempitsky and Ivanov's model since it is easy and fast to optimize and provides good results, as we will show later.

[47] We already encountered the concept of *regularization* in the context of multi-view stereo. Without it, the optimization only puts constraints on seam vertices and non-seam vertices can take arbitrary values. Obviously, we do not want the non-seam vertices' values to oscillate wildly. Among all solutions fulfilling the main objective – minimizing seam differences – the regularization makes the optimization prefer the "simplest", most "well-behaved" solution.

times during optimization and fit into memory (the table has $\mathcal{O}(\#\text{faces} \cdot \#\text{views})$ entries and is very sparse).

3.3.2. View Selection

Our view selection follows the structure of the base method; we obtain a labeling such as the one in Figure 3.4 (left) by optimizing Equation (3.1) with a Markov random field solver. However, we replace the base method's data and smoothness terms and augment the data term with a photo-consistency check.

3.3.2.1. Data Term

For the reasons described in Section 3.2 we choose the data term of Gal et al. [2010]:

$$E_{\text{data}}(F_i, \ell_i) = -\int\limits_{\phi(F_i, I_{\ell_i})} \|\nabla(I_{\ell_i}(p))\|_2 \, \mathrm{d}p. \tag{3.3}$$

With a Sobel operator we compute the gradient magnitude image $\|\nabla(I_{\ell_i})\|_2$ of the image I_{ℓ_i} that corresponds to the label ℓ_i. Further, we compute the projection $\phi(F_i, I_{\ell_i})$ of the face F_i in I_{ℓ_i} and sum over all pixels of the gradient magnitude image within this projection. If the projection is smaller than one pixel we sample the gradient magnitude image at the projection's centroid and multiply it with the projection area.

The data term's preference for large gradient magnitudes entails a problem that Gal et al. do not account for because it does not occur in their controlled datasets: Views containing occluders such as pedestrians that have not been reconstructed and thus cannot be detected by the visibility check should not be chosen for texturing the faces behind the occluder. Unfortunately, the gradient magnitude term frequently chooses occluders because they often feature a larger gradient magnitude than their background, *e.g.*, leaves in front of a uniform wall. We therefore introduce a step to ensure the texture's photo-consistency.

3.3.2.2. Photo-Consistency Check

We assume that for a specific face the majority of views see the correct color. A minority may see wrong colors (*i.e.*, occluders) and those are much less correlated. Based on this, Sinha et al. [2008] and Grammatikopoulos et al. [2007] use mean or median colors to reject inconsistent views. This is insufficient, as we show in Section 3.4.2.1. Instead we use a modified mean-shift algorithm:

1. Compute the face projection's mean color $\boldsymbol{c}_i$ (which is a 3-vector with one entry per color channel) for each view i in which the face is visible.
2. Put all views seeing the face into an inlier list.
3. Compute mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ of all inliers' mean colors $\boldsymbol{c}_i$.

4. Evaluate a 3D Gaussian $\exp(-\frac{1}{2}(\boldsymbol{c}_i - \boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{c}_i - \boldsymbol{\mu}))$ for each view in which the face is visible.
5. Clear the inlier list and insert all views whose function value is above $6 \cdot 10^{-3}$.
6. Repeat steps 3.–5. for 10 iterations or until all entries of $\boldsymbol{\Sigma}$ drop below 10^{-5}, the inversion of $\boldsymbol{\Sigma}$ becomes unstable, or the number of inliers drops below 4.

We obtain a list of photo-consistent views for each face and multiply a penalty on all other views' data terms to prevent their selection.

Using the median instead of the mean does not work on very small query sets because for 3D vectors, the marginal median is usually not a member of the query set, so that too many views are purged. Not shifting the mean does not work in practice because the initial mean is often far away from the inliers' mean (see Section 3.4.2.1). Sinha et al. [2008] therefore additionally allow the user to interactively mark regions that should not be used for texturing, a step which we explicitly want to avoid.

3.3.2.3. Smoothness Term

As discussed above, Lempitsky and Ivanov's smoothness term is a major performance bottleneck and counteracts our data term's preference for close-up views. We propose a smoothness term based on the Potts model: $E_{\text{smooth}}(F_i, F_j, \ell_i, \ell_j) = [\ell_i \neq \ell_j]$ ($[\cdot]$ is the Iverson bracket). This also prefers compact patches without favoring distant views and is extremely fast to compute.

Optimization: For optimizing Equation (3.1), the base algorithm [Lempitsky and Ivanov 2007] and our own basic implementation use GCO [Boykov et al. 2001]. Further, we developed a massively parallel MRF solver that is especially suited for very large MRFs with large label sets. We will describe it and analyze its performance in Chapter 4.

3.3.3. Color Adjustment

Models obtained from the view selection phase (*e.g.*, Figure 3.4 (right)) contain many color discontinuities between patches. These need to be adjusted to minimize seam visibility. We use an improved version of the base method's global adjustment, followed by a local adjustment with Poisson editing [Pérez et al. 2003].

3.3.3.1. Global Adjustment

A serious problem with Lempitsky and Ivanov's color adjustment is that $f_{v_{i,\text{left}}}$ and $f_{v_{i,\text{right}}}$ in Equation (3.2) are only evaluated at a single location: the vertex v's projection into the two images adjacent to the seam. If there are even small registration errors (which there always are), both projections do not correspond to exactly the same spot on the real object. Moreover, if both images have a different scale, the pixels corresponding to $f_{v_{i,\text{left}}}$ and $f_{v_{i,\text{right}}}$ span a different footprint on the surface and their colors may be very different. This may be irrelevant in controlled lab datasets,

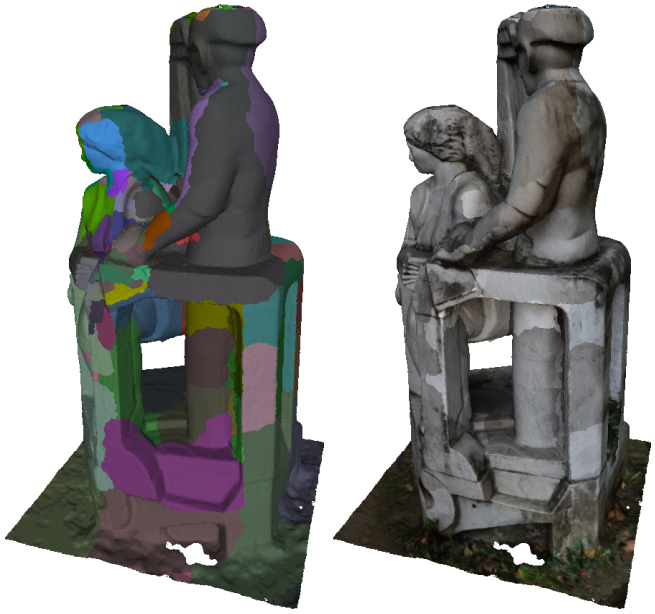


Figure 3.4.: *Left:* A mesh's labeling. Each color represents a different label, *i.e.*, input image. *Right:* The textured result with visible luminance differences between patches.

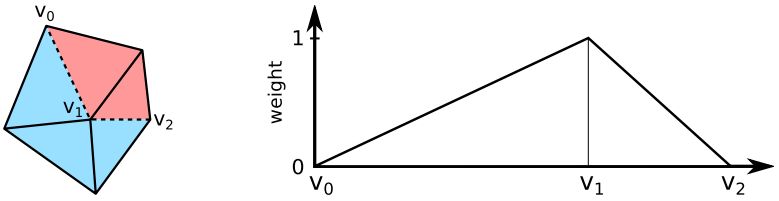


Figure 3.5.: *Left:* A mesh. Vertex v_1 is adjacent to both texture patches (red and blue). Its color is looked up as a weighted average over samples on the edges $\overline{v_0v_1}$ and $\overline{v_1v_2}$. *Right:* Sample weights transition from 1 to 0 as distance to $\check{1}$ grows.

but in realistic multi-view stereo datasets the lookups of effectively different points or with different footprints mislead the global adjustment.

3.3.3.2. Color Lookup Support Region

We alleviate this problem by looking up a vertex's color value not just at the vertex projection but along all adjacent seam edges, as illustrated by Figure 3.5: Vertex v_1 is on the seam between the red and the blue patch. We evaluate its color in the red patch, $f_{v_{1,\text{red}}}$, by averaging color samples from the red image along the two edges $\overline{v_0v_1}$ and $\overline{v_1v_2}$. On each edge we take twice as many samples as the edge length in pixels. When averaging the samples, we weight them according to Figure 3.5 (right): The weight is 1 on v_1 and decreases linearly with a sample's distance to v_1. (The reasoning behind this is that after optimizing Equation (3.2), the computed correction $g_{v_{1,\text{red}}}$ is applied to the texels using barycentric coordinates.

Along the seam the barycentric coordinates form the transition from 1 to 0.) We obtain average colors for the edges $\overline{v_0v_1}$ and $\overline{v_1v_2}$, which we average weighted with the edge lengths to obtain $f_{v_{1,\text{red}}}$. Similarly, we obtain $f_{v_{1,\text{blue}}}$ and insert both into Equation (3.2).

For optimization, Equation (3.2) can now be written in matrix form:

$$\min_{\boldsymbol{g}} \sum_{v_i \in \ldots} \left((f_{v_{i,\text{left}}} + g_{v_{i,\text{left}}}) - (f_{v_{i,\text{right}}} + g_{v_{i,\text{right}}}) \right)^2 + \lambda^2 \sum_{\{v_i,v_j\} \in \ldots} \left(g_{v_i} - g_{v_j} \right)^2 \tag{3.4}$$

$$= \min_{\boldsymbol{g}} \sum_{v_i \in \ldots} \left(g_{v_{i,\text{left}}} - g_{v_{i,\text{right}}} - (f_{v_{i,\text{right}}} - f_{v_{i,\text{left}}}) \right)^2 + \lambda^2 \sum_{\{v_i,v_j\} \in \ldots} \left(g_{v_i} - g_{v_j} \right)^2 \tag{3.5}$$

$$= \min_{\boldsymbol{g}} \left(\|\boldsymbol{A}\boldsymbol{g} - \boldsymbol{f}\|_2^2 + \|\lambda \boldsymbol{\Gamma} \boldsymbol{g}\|_2^2 \right), \tag{3.6}$$

where

$$\boldsymbol{f} = \begin{pmatrix} f_{v_{i,\text{right}}} - f_{v_{i,\text{left}}} \\ \vdots \\ f_{v_{k,\text{right}}} - f_{v_{k,\text{left}}} \end{pmatrix} \tag{3.7}$$

is a vector that stacks all $f_{v_{i,\text{right}}} - f_{v_{i,\text{left}}}$ from Equation (3.5),

$$\boldsymbol{g} = (g_1, \ldots, g_n)^\mathsf{T} \tag{3.8}$$

is a vector of the adjustments for all vertices, and

$$\boldsymbol{A} = \overbrace{\begin{pmatrix} 1 & & -1 & & \\ & -1 & & -1 & 1 \end{pmatrix}}^{\#\text{columns}=\#\text{vertices}} \Bigg\}_{\#\text{rows}=\#\text{seam vertices}} \tag{3.9}$$

$$\text{and } \boldsymbol{\Gamma} = \overbrace{\begin{pmatrix} & & 1 & -1 & \\ -1 & & & & 1 \\ 1 & -1 & & & \end{pmatrix}}^{\#\text{columns}=\#\text{vertices}} \Bigg\}_{\#\text{rows}=\#\text{pairs of adjacent vertices}} \tag{3.10}$$

are sparse matrices containing one $+1$ entry and one -1 entry per row to pick the correct $g_{v_{i,\text{left}}}$, $g_{v_{i,\text{right}}}$, g_{v_i}, and g_{v_j}, respectively, from $\boldsymbol{g}$. Equation (3.6) is called Tikhonov regularization (or ridge regression in machine learning). The so-called Tikhonov matrix $\boldsymbol{\Gamma}$ determines how the solution is regularized (in our case: adjustments should vary smoothly between adjacent vertices). With λ we can control the strength of this regularization. Multiplying out Equation (3.6) yields

$$\min_{\boldsymbol{g}} \left(\|\boldsymbol{A}\boldsymbol{g} - \boldsymbol{f}\|_2^2 + \|\lambda \boldsymbol{\Gamma} \boldsymbol{g}\|_2^2 \right)$$

$$= \min_{\boldsymbol{g}} \left(\boldsymbol{g}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{g} - \boldsymbol{g}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{f} - \boldsymbol{f}^\mathsf{T}\boldsymbol{A}\boldsymbol{g} + \boldsymbol{f}^\mathsf{T}\boldsymbol{f} + \lambda^2 \boldsymbol{g}^\mathsf{T}\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{g} \right) \tag{3.11}$$

$$= \min_{\boldsymbol{g}} \left(\boldsymbol{g}^\mathsf{T}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2 \boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma})\boldsymbol{g} - 2\boldsymbol{f}^\mathsf{T}\boldsymbol{A}\boldsymbol{g} + \boldsymbol{f}^\mathsf{T}\boldsymbol{f} \right), \tag{3.12}$$

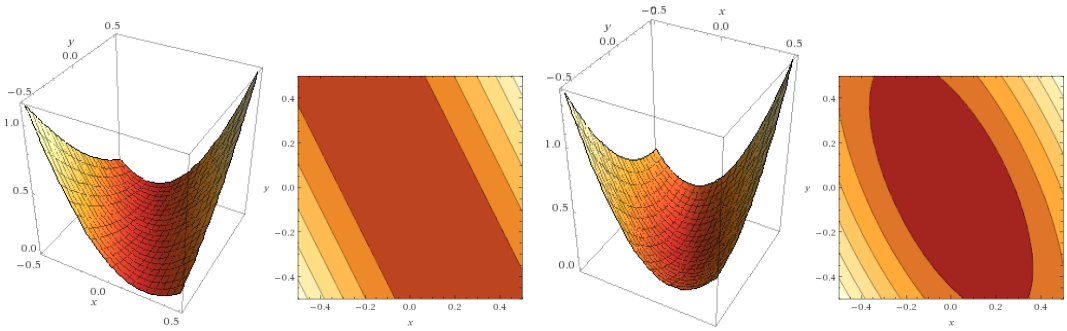


Figure 3.6.: *1st and 2nd image:* Graph and contour plot of the quadratic form $\boldsymbol{x}^\mathsf{T} \left(\begin{smallmatrix} 2 & 1 \\ 1 & 0.5 \end{smallmatrix}\right) \boldsymbol{x}$. Since the matrix is singular, it has a horizontal valley of minima. *3rd and 4th image:* Graph and contour plot of $\boldsymbol{x}^\mathsf{T} \left(\begin{smallmatrix} 2 & 1 \\ 1 & 0.5 \end{smallmatrix}\right) \boldsymbol{x} + 0.5\boldsymbol{x}^\mathsf{T}\boldsymbol{x}$. The regularization term $0.5\boldsymbol{x}^\mathsf{T}\boldsymbol{x}$ makes the minimum at $\boldsymbol{x} = (0,0)^\mathsf{T}$ unique.

which is a quadratic form in $\boldsymbol{g}$. Due to the size and sparsity of $\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma}$, it does not make sense to compute the solution $\hat{\boldsymbol{g}} = \left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma}\right)^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{f}$ explicitly.[48] This should be done with an iterative solver. Since $\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma}$ is very sparse, symmetric, and positive semidefinite,[49] we can use the conjugate gradient (CG) method [Shewchuk 1994]. We use Eigen's [Guennebaud et al. 2010] CG implementation, initialize it with $\boldsymbol{g}_{(0)} = (0,\ldots,0)^\mathsf{T}$ and stop it when $\frac{\|\boldsymbol{r}\|_2}{\left\|\boldsymbol{A}^\mathsf{T}\boldsymbol{f}\right\|_2} < 10^{-5}$ ($\boldsymbol{r}$ is the residual). CG requires at most n steps for solving, where n is the dimensionality $\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma}$, but in practice it typically requires less than 200 iterations for our texturing problems even for very large datasets.

The solution to Equation (3.12) is actually not unique: We can add a constant to each entry in the solution vector $\hat{\boldsymbol{g}}$, *i.e.*, $\hat{\boldsymbol{g}}+(c,\ldots,c)^\mathsf{T}$, and still obtain an optimum.[50] This means that $\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma}$ is singular and the associated quadratic form has a valley at its bottom (see a very simplified example in the left two images of Figure 3.6). This is fine for CG and it will just converge to any point in this valley. In our application, however, we need to take care that we do not obtain negative or arbitrarily large positive color values. Lempitsky and Ivanov [2007] solve this by computing the mean $\bar{g}$ of all g_i in the found solution $\hat{\boldsymbol{g}}$ and subtracting it from all entries: $\hat{\boldsymbol{g}}_{\text{normalized}} = \hat{\boldsymbol{g}} - (\bar{g},\ldots,\bar{g})^\mathsf{T}$. Thereby, the adjusted model differs as little as possible from the unadjusted model.

A more elegant solution is to regularize the length of the solution vector:

$$\min_{\boldsymbol{g}} \left(\|\boldsymbol{A}\boldsymbol{g} - \boldsymbol{f}\|_2^2 + \|\lambda\boldsymbol{\Gamma}\boldsymbol{g}\|_2^2 + \varepsilon \|\boldsymbol{g}\|_2^2 \right). \tag{3.13}$$

[48] Also, $\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma}$ is singular as we will discuss later.

[49] $\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma}$ is positive semidefinite because $\forall \boldsymbol{z}$: $\boldsymbol{z}^\mathsf{T}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda^2\boldsymbol{\Gamma}^\mathsf{T}\boldsymbol{\Gamma})\boldsymbol{z} = \underbrace{\|\boldsymbol{A}\boldsymbol{z}\|_2^2}_{\geq 0} + \lambda^2 \underbrace{\|\boldsymbol{\Gamma}\boldsymbol{z}\|_2^2}_{\geq 0} \geq 0$.

[50] $\|\boldsymbol{A}\boldsymbol{g} - \boldsymbol{f}\|_2^2$ in Equation (3.6) – the squared difference between a seam vertex's color in the left and right texture – stays the same if we add a constant to the left and right color. Likewise, $\|\lambda\boldsymbol{\Gamma}\boldsymbol{g}\|_2^2$ stays the same if we add a constant to both vertices of a pair connected by $\boldsymbol{\Gamma}$.

The term $\varepsilon \|\boldsymbol{g}\|_2^2$ "bends" the quadratic form's solution valley such that only one point – the one closest to the origin $(0, \ldots, 0)^\mathsf{T}$ – is a minimum of the objective function. This is demonstrated on a simplified example in the right half of Figure 3.6 where we chose $\varepsilon = 0.5$ to be relatively large for illustration purposes. In practice we need to choose ε small enough that the effect of $\varepsilon \|\boldsymbol{g}\|_2^2$ is much weaker than the effect of $\|\lambda \boldsymbol{\Gamma} \boldsymbol{g}\|_2^2$ but big enough that it is not eliminated due to numerical (in)accuracy and that the slope of the valley is not so small that CG decides to stop.

This regularization has another advantage: Large datasets often consist of multiple sub-meshes that are not connected with each other. Our argument above for adding a constant to the adjustments of all vertices actually works for each component *independently of all other components* because $\boldsymbol{\Gamma}$ never couples two vertices of different components. In order to enforce solution uniqueness with Lempitsky and Ivanov's method and keep all color values in an acceptable range, we have to first find all mesh components, compute each component's $\bar{g}$, and compute $\hat{\boldsymbol{g}} - (\bar{g}, \ldots, \bar{g})^\mathsf{T}$ for each vertex in a component.

Due to, *e.g.*, automatic white balancing, different camera response curves, and different light colors between images (noon vs. sunset vs. artificial light), it is not sufficient to only optimize the luminance channel and we optimize all three channels in parallel.

3.3.3.3. Local Adjustment: Poisson Editing

Even with the above support regions, Lempitsky and Ivanov's global adjustment does not eliminate all visible seams, as exemplified in Figure 3.14 (bottom row, center). Thus, after global adjustment we additionally perform local Poisson image editing [Pérez et al. 2003]. Gal et al. [2010] do this as well, but in a way that makes the computation prohibitively expensive on realistic datasets: They Poisson edit complete texture patches, which results in huge equation systems with $> 10^6$ variables for the largest patches in our datasets.

We thus restrict the Poisson editing of a patch to a 20 pixel wide border strip (shown in light blue in the left of Figure 3.7). If the patch is too small, we omit the

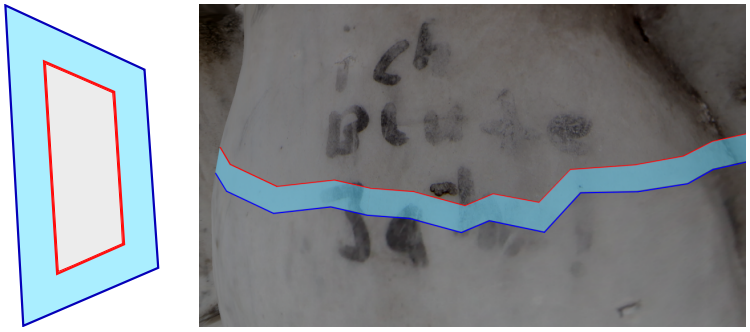


Figure 3.7.: *Left:* Abstract visualization of a texture patch whose border strip (light blue) has an outer (dark blue) and inner rim (red). *Right:* Border strip in an actual reconstruction.

inner rim. We use this strip's outer rim (dark blue in Figure 3.7) and inner rim (red in Figure 3.7) as boundary conditions for the Poisson equations:

$$\begin{pmatrix} \ddots & & & & & & & & & & & & & & \\ & \ddots & & & & & & & & & & & & & \\ & & 1 & & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & & \\ & & & & \ddots & & & & & & & & & & \\ & & & & & \ddots & & & & & & & & & \\ & & & & & & \ddots & & & & & & & & \\ & & -1 & & & & -1 & 4 & -1 & & & & -1 & & \\ & & & -1 & & & & -1 & 4 & -1 & & & & -1 & \\ & & & & -1 & & & & -1 & 4 & -1 & & & & -1 \\ & & & & & & & & & & \ddots & & & & \\ & & & & & & & & & & & \ddots & & & \\ & & & & & & & & & & & & \ddots & & \\ & & & & & & & & & & & & & 1 & \\ & & & & & & & & & & & & & & 1 \\ & & & & & & & & & & & & & & & \ddots \\ & & & & & & & & & & & & & & & & \ddots \end{pmatrix} \cdot \begin{pmatrix} \\ \\ \\

Table 3.1.: Summary of the datasets used in the following experiments.

dataset	# views = # MRF labels	# mesh faces = # MRF vertices	image resolution
Kopernikus Statue	334	4.9 million	5616×3744
Castle Ruin	287	20.3 million	5616×3744
Citywall-100	561	8.2 million	2000×1500
Citywall-40	561	2.4 million	2000×1500
Citywall-20	561	0.7 million	2000×1500
Reader-100	539	12.5 million	5400×3600
Reader-40	539	5.7 million	5400×3600
Reader-20	539	1.2 million	5400×3600

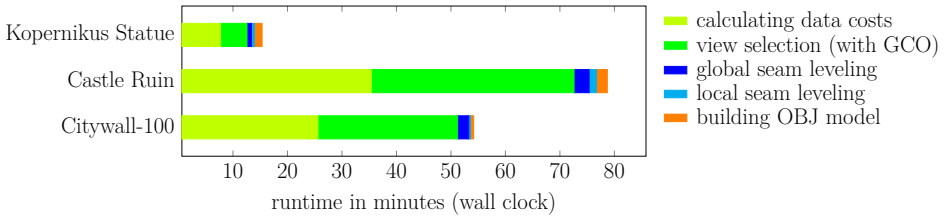


Figure 3.8.: Runtime of our texturing algorithm on various datasets with graph cut optimization (GCO) used for the view selection MRF.

– the latter two in three different sizes each. Table 3.1 summarizes their properties.

3.4.1. Runtime

We first analyze the runtime of our algorithm on a machine with two 8-core Xeon E5-2650v2 CPUs and 128 GB of memory. Figure 3.8 shows runtime graphs for the Kopernikus Statue, the Castle Ruin and the Citywall-100. Even our largest dataset, the Castle Ruin (see rendered results in Figure 3.9), can be textured in less than 80 min, which is in stark contrast to other methods which may yield better results but require a tremendous amount of computation. *E.g.*, Goldluecke et al. [2014] compute for several hours (partially on the GPU) to texture small lab-sized models with 36 views within a super-resolution framework. Our algorithm manages to texture large-scale real-world datasets in acceptable time.

For all three datasets, our algorithm's main computational bottlenecks are the data cost precomputation whose complexity is linear in the number of views and the number of pixels per view, and the graph cut-based view selection which is linear in the number of mesh faces and the number of views. By altering the above datasets' properties, *e.g.*, by simplifying the mesh, we found that linear complexity fits closely in practice. The data term computation is already fully parallelized on the CPU[51]

[51] It has further been parallelized to GPUs in the course of a student project by Patrick Seemann and Nils Möhrle. For very large datasets ($>10^7$ triangles) the speedups are $\sim$10 for a GeForce GTX

Figure 3.9.: Texturing results of the Castle Ruin dataset.

but the graph cut cannot be parallelized trivially. In Chapter 4 we present a solution which brings down the optimization times on our texturing datasets from hours to minutes, which is unrivaled by solvers from prior work.

3.4.2. Qualitative Evaluation

In this section, we evaluate the individual components of our approach qualitatively and compare them to prior work.

3.4.2.1. View Selection Data Term

As argued earlier, the view selection's data term must take the input images' content and not only their geometric configuration into account. Allène et al.'s [2008] data term computes a face's projection area and thus selects the image in the top right of Figure 3.10 to texture the Kopernikus Statue's arm (yielding the result in 3.10's top left) even though this image is not focused on the arm but on its background. Gal et al.'s [2010] data term instead favors large gradient magnitudes and large projection areas. It thus uses the image in the bottom right of Figure 3.10 for the arm, yielding a much crisper result as shown in 3.10's bottom left.

One weakness of Gal et al.'s data term is artifacts with a strong gradient magnitude. This is even more pronounced in datasets with little surface texture detail where the gradient magnitude of undesired image content can easily be higher than that of "good" image content. One such example is the Middlebury Dino [Seitz et al.

Titan X over the double 8-core Xeon E5-2650v2 CPU mentioned above. The main bottleneck is the ray intersection visibility check rather than the data term computation (Equation (3.3)). Therefore, speedups are higher for convex objects, where visibility can mostly be determined through back-face and view frustum culling, than for geometrically complex datasets. On small, complex datasets the speedup can be as low as 2.

Figure 3.10.: *Top row:* Kopernikus Statue's arm textured with the data term of Allène et al. [2008] and a close-up of the input image that was picked to texture the arm where the arm itself is out of focus. *Bottom row:* The arm textured with the data term of Gal et al. [2010] and a close-up of the picked input image with the arm region being in focus.

2006] which is based on a uniform white plaster figurine. All views in that dataset have approximately the same distance to the object and out-of-focus blur is not an issue. Thus, Gal et al.'s data term accounts for effects that do not occur in this dataset. We found that the data term instead prefers regions with artifacts such as pixels in the vicinity of occlusion boundaries, mistaking them for surface detail; see Figure 3.11. The photo-consistency check (which we analyze in the following section) partially alleviates this effect but only to the point that we see in Figure 3.11 (meaning that without the check the model contains even more such artifacts).

3.4.2.2. Photo-Consistency Check

Similarly, Gal et al.'s data term frequently chooses unreconstructed occluders to texture the model behind the occluder (see Figure 3.12, top right) because many occluders contain more high-frequency image content than the "correct" texture.

In Figure 3.12 (bottom left), we show the photo-consistency check at work for one example face. This face has many outliers (red), *i.e.*, views seeing an occluder

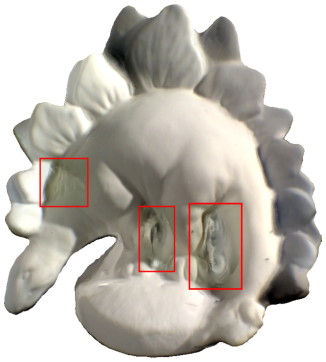
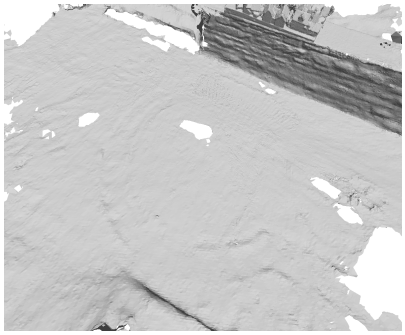
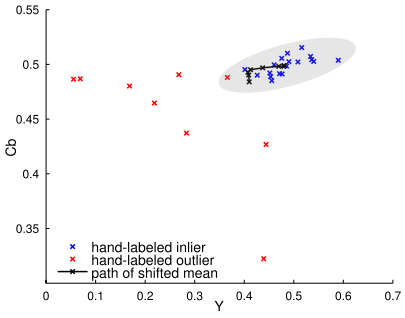


Figure 3.11.: Furukawa and Ponce's [2010] Middlebury Dino Ring reconstruction textured with our framework (artifacts highlighted in red).



(a) Input mesh

(b) Pedestrian used as texture for the ground



(c) Photo-consistency check visualization for a single triangle. Red and blue crosses are the triangle's mean color (here we show only the Y and C_b component) in different views.

(d) Mesh textured with photo-consistency check

Figure 3.12.: Occluder removal through our photo-consistency check.

instead of the correct face content. The gray ellipse marks all views that our check classifies as inliers after convergence. Only one view is misclassified. Note the black path that the shifted mean takes: Its starting point on the path's bottom left is the algorithm's initial mean. Sinha et al. [2008] and Grammatikopoulos et al. [2007] use it without iterating in mean-shift fashion. It is, however, a bit off from the true inlier mean and using a fixed window around it would result in several false positives and

Figure 3.13.: *Left:* The Kopernikus Statue's shoulder with Lempitsky and Ivanov's smoothness term. *Right:* Resulting model with the Potts model smoothness term.

negatives. Using all views' covariance matrix would classify all views except for the bottommost data point as inliers. Our approach only misclassifies the one outlier closest to the inlier set and successfully removes the pedestrian (Figure 3.12, bottom right).

Our proposed photo-consistency check may fail, *e.g.*, if a face is seen by very few views. In this case there is little image evidence for what is an inlier or an outlier and the classification becomes inaccurate.

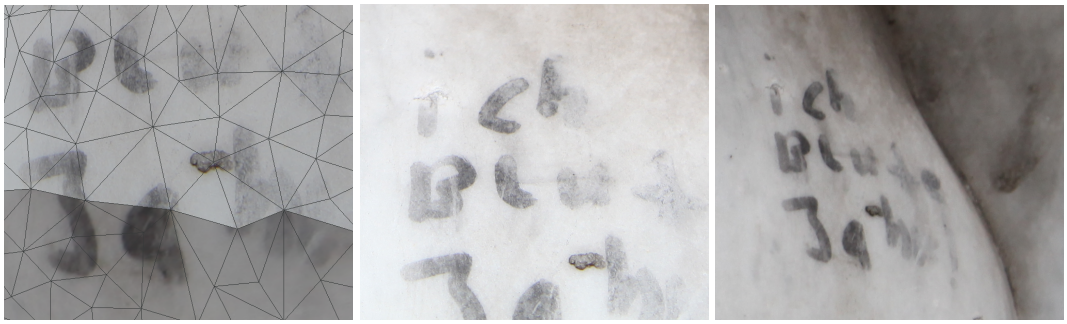
3.4.2.3. View Selection Smoothness Term

As view selection smoothness term, Lempitsky and Ivanov use the error on a seam integrated along the seam. In the results, blurry texture patches selected from distant views occur frequently since blur makes seams less noticeable (Figure 3.13 (left)). The Potts model yields a considerably sharper result (Figure 3.13 (right)).

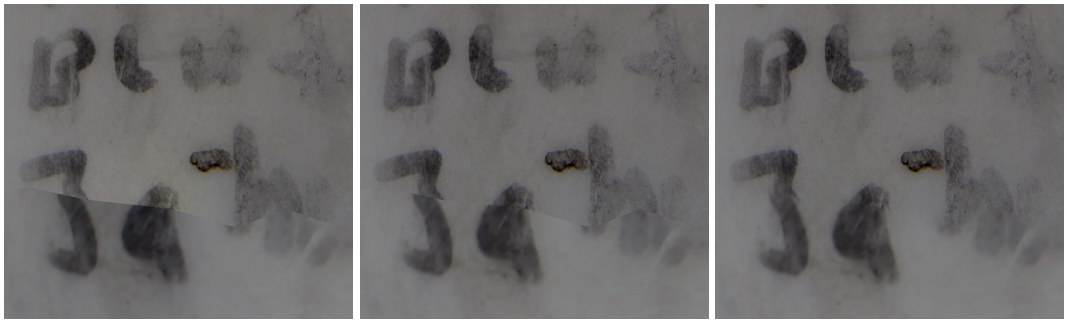
3.4.2.4. Color Adjustment

Lempitsky and Ivanov's global color adjustment operates on a per-vertex base. It fails if a seam vertex looked up in the images to the left and right of the seam does not project onto the exact same 3D region. Such image correspondence errors occur for various reasons, *e.g.*, camera misalignment, 3D reconstruction inaccuracies, or images with different pixel footprint. In Figure 3.14 (top left), the vertex on the right side of the letter 'J' projects onto the letter itself to the seam's top and onto the background to the seam's bottom. The result after global adjustment (bottom left) exhibits a strong color artifact. We alleviate this using support regions during sampling (bottom center). The remaining visible seams are fixed with Poisson editing as shown in Figure 3.14 (bottom right). Poisson editing does not correct texture alignment problems but disguises most seams well so that they are virtually invisible unless the model is inspected from a very close distance.

Color adjustment problems frequently occur in Lempitsky and Ivanov's method when the mesh resolution is much lower than the image resolution, because then the



(a) Unadjusted mesh with superimposed wireframe **(b)** Detail of the input image used for the upper patch **(c)** Detail of the input image used for the lower patch



(d) Artifact from global color adjustment caused by texture misalignment **(e)** Result after global adjustment with support regions **(f)** Result after Poisson editing

Figure 3.14.: Results of global adjustment and Poisson editing.

color of one single pixel (the vertex's projection) is unrepresentative for the vertex's neighborhood's true color. Figure 3.15 shows an example: The result without support regions and Poisson editing (bottom left) exhibits strong color artifacts (white arrows) when compared to a photograph of the area (top). The result with support region and Poisson editing (bottom right) has a significantly better quality.

3.5. Conclusions and Future Work

Applying textures to reconstructed models is one of the keys to realism. Surprisingly, this topic has been neglected in recent years and the state of the art is still to reconstruct models with per-vertex colors. We therefore presented the first comprehensive texturing framework for large real-world 3D reconstructions from registered images. Typical effects that occur frequently in these datasets are camera parameter and geometry inaccuracies, lighting and exposure variation, image scale variation, out-of-focus blur, and unreconstructed occluders. Based on existing work, we made several key changes that handle the mentioned challenges automatically and efficiently. Further, we demonstrated the entailing improvements in the textured

(a) One of the input images

(b) Citywall reconstruction color adjusted w/o support region and w/o Poisson editing

(c) The same reconstruction color adjusted with support region and Poisson editing

Figure 3.15.: Influence of support regions and Poisson editing. (Best viewed on screen.)

models. Large-scale, real-world geometry reconstructions can now be enriched with high-quality textures which will significantly increase the realism of reconstructed models. Our framework allows texturing meshes with more than 20 million faces using close to 300 images in less than 80 minutes.

As future work we see a lot of remaining topics. Most importantly, we are convinced that future texturing frameworks need to address *inaccurate, incomplete or very unstructured data*: Inaccurate camera parameters or geometry can, *e.g.*, be handled by displacing the geometry or the input images, but this needs to be done such that it explains the input data best (in contrast to Gal et al. [2010] who minimize seam visibility without maximizing agreement with the input images) and that it can be optimized efficiently (in contrast to Goldluecke et al. [2014]). Further, texturing frameworks need to assign a texture even in regions where none has been observed. Surface reconstruction methods interpolate surfaces in regions with little or no support from input views and the resulting surfaces also need to be textured, *e.g.*, with texture synthesis/transfer. Finally, texturing frameworks need to be robust enough to handle community photo collection datasets. Their unstructured and wildly varying properties are already addressed in various works on structure from motion and multi-view stereo [Snavely et al. 2006, Agarwal et al. 2009, Frahm et al. 2010, Heinly et al. 2015, Furukawa et al. 2010, Schönberger et al. 2016]. In contrast, our texturing framework struggles with them (Figure 9.3 shows an example, which was, however, by far not the worst among those that we encountered in our experiments). The key insight from structure from motion and multi-view stereo is that community photo collections contain so much redundant data that presumably erroneous or noisy data can generously be discarded. Thus, we must distinguish

between “good” and “bad” data and generously err on the side of discarding good data. In that context, our photo-consistency model (Section 3.3.2.2) is probably too simple and needs to be robustified for situations with many drastic outliers.

4. Increasing Texturing Speed

When looking at Figure 3.8, we can observe that our texturing approach's runtime is dominated by two bottlenecks: Data cost precomputation and solving the view selection's Markov random field (MRF). From a scientific point of view, parallelizing the data cost computation is rather uninteresting since the computation of each data cost table entry is independent of the computation of all other entries and therefore trivially parallelizable. It has, in fact, already been fully parallelized on the CPU and GPU. The more interesting avenue of research is the parallelization of the MRF optimization, which is what we will look into in the course of this chapter.

In Section 4.1 we very briefly review the properties of some existing MRF solvers and the implications on our view selection problem. In Section 4.2 we then sketch our massively parallel MRF solver "mapMAP", and in Section 4.3 we demonstrate its runtime improvements in our texturing application.

4.1. Existing MRF Solvers

We already discussed α-expansion in Section 2.4.3. Here we will first shed light on the implications of using it for solving our texturing MRFs (Section 3.3.2).

Looking at the datasets "Citywall-100" and "Reader-100" that we [Thuerck et al. 2016] analyzed, we can see that, while the full label set contains 561 and 539 labels, respectively, each vertex has on average only 81/56 *feasible* labels, which is 14 %/10 % of the full label set. This comes as no surprise, since on realistic multi-view reconstruction datasets each input image sees only a fraction of the entire scene. In each iteration of α-expansion's inner loop, the graph cut call (Line 4 in Listing 2.2) has to construct a graph larger than the original MRF on which it can then solve the two-label problem. While a label that is infeasible for many vertices results in a smaller graph than a label that is feasible everywhere, a large graph still has to be constructed and solved in each and every iteration. On the texturing datasets there are typically $\sim$4000 such graph cut calls and each of them is inefficient because most vertices cannot change their label to the new label α.

Another problem with α-expansion is that it works only on one CPU core and therefore does not leverage the power of modern multi-core CPUs or many-core systems such as GPUs. There are algorithms that replace the graph cut algorithm within α-expansion with a parallel variant. However, some are limited to regular MRF topologies such as 4-connected 2D grids [Jamriska et al. 2012] and are therefore unsuitable for our irregularly structured texturing MRFs. And even for those that do support irregular topologies, it turns out that parallel graph cut does not scale

very well, especially on many cores. For the work of Shekhovtsov and Hlavác [2013], we shall see in Section 4.3.1 that it performs even worse than serial α-expansion.

The other class of solvers that we discussed in the background chapter (Section 2.4.4), are solvers that reduce the MRF's topology. Chen and Koltun [2014] are restricted to grid-shaped topologies. They claim they only made this restriction to simplify their paper's exposition and that it "can be generalized beyond this setting" [Chen and Koltun 2014, beginning of page 2]. We claim that this is not trivial. The solver of Veksler [2005] does work on arbitrarily shaped MRF topologies by sampling spanning trees on the topology and solving these trees with dynamic programming, which can even be parallelized. Since the spanning trees cover every vertex in the MRF, many edges between vertices need to be cut in order to obtain a tree, the smoothness terms corresponding to the cut edges are not added to the energy computation, the tree's energy becomes unrelated to the original MRF's energy, and as a result applying the solution of the reduced MRF to the original MRF may actually increase the energy of the original MRF. Another issue with that algorithm is that all vertices are being optimized and therefore there are no fixed vertices that can "remember" a previous iteration's solution. Thus, even if we used it in an iterative way, each iteration would be a fresh restart that always gives the same guarantee-free energy.

In contrast to, *e.g.*, α-expansion, Veksler's and Chen and Koltun's solvers can both handle arbitrary smoothness cost types and can relatively easily be parallelized. However, Veksler's potential energy increase and Chen and Koltun's restriction to grid-shaped MRFs are certainly undesirable. We present a generalization to arbitrary MRF topologies that gives an energy decrease guarantee and can be parallelized in a very fine-grained manner.

4.2. mapMAP (Minimal Assumption, High-Performance MAP solver)

To present our MRF solver, we first introduce a bit of notation. As mentioned earlier, an MRF is a graph $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of vertices and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is a set of edges. We then try to assign a label $\ell \in \mathcal{L}$ to each $v \in \mathcal{V}$ such that the following expression is minimized:

$$\underset{(\ell_{v_1},\ldots,\ell_{v_n}) \in \mathcal{L}^n}{\arg\min} \sum_{v_i \in \mathcal{V}} d(\ell_{o_i} - \ell_{v_i}) + \lambda \sum_{\{v_i,v_j\} \in \mathcal{E}} s(\ell_{v_i} - \ell_{v_j}) \tag{4.1}$$

with so-called *data costs* d and *smoothness costs* s. A block-coordinate descent (BCD) algorithm selects a subset of all MRF vertices called *coordinates* $\mathcal{C}$ and solves a reduced problem based on $\mathcal{C}$. Thus, the reduced problem is an MRF

$$(\mathcal{C}, \mathcal{E}_{\mathcal{C},\mathcal{C}} \cup \mathcal{E}_{\mathcal{C},\mathcal{F}}), \tag{4.2}$$

with

$$\mathcal{C} \subseteq \mathcal{V}, \tag{4.3}$$

$$\mathcal{F} = \mathcal{V} \setminus \mathcal{C}, \tag{4.4}$$
$$\mathcal{E}_{\mathcal{C},\mathcal{C}} \subseteq \mathcal{E} \cap (\mathcal{C} \underline{\times} \mathcal{C}), \tag{4.5}$$
$$\mathcal{E}_{\mathcal{C},\mathcal{F}} \subseteq \mathcal{E} \cap (\mathcal{C} \underline{\times} \mathcal{F}) \tag{4.6}$$

(recall that we defined the set operator $\underline{\times}$ as Cartesian product that produces *unordered* pairs: $A \underline{\times} B = \{\{a, b\} : a \in A, b \in B\}$), and the objective function

$$\underset{(\ell_{c_1},\ldots,\ell_{c_m})\in\mathcal{L}^m \mid c_i\in\mathcal{C}}{\arg\min} \quad \sum_{v_i \in \mathcal{C}} d(\ell_{o_i} - \ell_{v_i}) + \lambda \cdot \sum_{\{v_i,v_j\} \in \mathcal{E}_{\mathcal{C},\mathcal{C}} \cup \mathcal{E}_{\mathcal{C},\mathcal{F}}} s(\ell_{v_i} - \ell_{v_j}). \tag{4.7}$$

We call $\mathcal{F}$ *fixed vertices*, $\mathcal{E}_{\mathcal{C},\mathcal{C}}$ *links*, and $\mathcal{E}_{\mathcal{C},\mathcal{F}}$ *dependencies*. Further, we call a reduced problem *compatible* with the original MRF, if its optimal solution applied to the original MRF decreases the original MRF's energy.

Our algorithm draws on ideas from Veksler and Chen and Koltun as we shall see now. What we learned from Veksler's algorithm is that, when selecting the reduced from the full MRF, we must not cut any links and must obey all dependencies. Otherwise, the reduced MRF becomes incompatible with the full MRF.[52] Therefore,

$$\mathcal{E}_{\mathcal{C},\mathcal{C}} = \mathcal{E} \cap (\mathcal{C} \underline{\times} \mathcal{C}) \text{ (as opposed to } \mathcal{E}_{\mathcal{C},\mathcal{C}} \subsetneq \mathcal{E} \cap (\mathcal{C} \underline{\times} \mathcal{C})), \tag{4.8}$$
$$\text{and } \mathcal{E}_{\mathcal{C},\mathcal{F}} = \mathcal{E} \cap (\mathcal{C} \underline{\times} \mathcal{F}) \text{ (as opposed to } \mathcal{E}_{\mathcal{C},\mathcal{F}} \subsetneq \mathcal{E} \cap (\mathcal{C} \underline{\times} \mathcal{F})). \tag{4.9}$$

This means that in a compatible MRF the contents of $\mathcal{F}$, $\mathcal{E}_{\mathcal{C},\mathcal{C}}$ and $\mathcal{E}_{\mathcal{C},\mathcal{F}}$ directly follow from the choice of $\mathcal{C}$. We therefore call

$$(\mathcal{C}, \mathcal{E}_{\mathcal{C},\mathcal{C}} \cup \mathcal{E}_{\mathcal{C},\mathcal{F}}), \tag{4.10}$$
$$\mathcal{C} \subseteq \mathcal{V}, \tag{4.11}$$
$$\mathcal{F} = \mathcal{V} \setminus \mathcal{C}, \tag{4.12}$$
$$\mathcal{E}_{\mathcal{C},\mathcal{C}} = \mathcal{E} \cap (\mathcal{C} \underline{\times} \mathcal{C}), \tag{4.13}$$
$$\mathcal{E}_{\mathcal{C},\mathcal{F}} = \mathcal{E} \cap (\mathcal{C} \underline{\times} \mathcal{F}) \tag{4.14}$$

an MRF that is *induced* by $\mathcal{C}$. Also, due to this direct relationship, we will in the following sometimes refer to $\mathcal{C}$ when actually meaning the MRF induced by $\mathcal{C}$ (*e.g.*, "$\mathcal{C}$ can be solved in polynomial time").

We further learned from Veksler's algorithm that trees are desirable because we can cover relatively large portions of an MRF with them, allowing for big energy improvements,[53] and still solve them in $O(|\mathcal{C}||\mathcal{L}|^2)$ using dynamic programming with no restrictions on the smoothness cost type. From our arguments above we learned that we cannot grow a tree to the point where it covers the whole MRF because we

[52] Thuerck et al. [2016] showed the inverse, *i.e.*, keeping all of them yields compatibility.

[53] Thuerck et al. [2016] showed that if $\mathcal{C}_1$ and $\mathcal{C}_2$ are two coordinate sets on the same MRF where $\mathcal{C}_1 \supsetneq \mathcal{C}_2$, then the MRF induced by $\mathcal{C}_1$ allows for bigger energy decreases than that induced by $\mathcal{C}_2$. This is intuitively clear because $\mathcal{C}_1$ and $\mathcal{C}_2$ are identical except for the additional coordinates that $\mathcal{C}_1$ has. Not keeping those fixed cannot make the solution worse. Even in the worst case the optimization can always just set the labels to the fixed values in the MRF induced by $\mathcal{C}_2$.

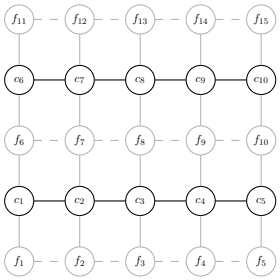


Figure 4.1.: Chen and Koltun's line separation: Coordinates and links are black, fixed vertices and dependencies are gray and fixed-fixed edges are dashed gray. Different coordinate lines can be solved in parallel without loosing compatibility, as long as any two coordinate lines are separated by one fixed line.

would then have to cut links and lose compatibility. Alternatively, if we do not cut them, the induced MRF will have cycles and we do not know how to solve general cyclic graphs in polynomial time. Thus, we must stop with tree growing before we run into this cycle-or-incompatible dilemma.

Although they do not grow trees, Chen and Koltun [2014] give us a good idea for a criterion for when to stop growing. Their reduced problems – lines of vertices, see an example in Figure 4.1 – are compatible with the original MRF.

4.2.1. Sampling Compatible Coordinate Trees

Chen's line separation idea (Figure 4.1) can be generalized: Coordinate trees are compatible with the original MRF if any two coordinates are either directly connected (c_1 and c_2) or separated by at least one fixed vertex (c_1 and c_6 are separated by f_6).

An algorithm to grow trees with this property would be: We start with one single random coordinate as root. For each fixed vertex we keep and update a counter of how many coordinate neighbors it has. We then pick a random fixed variable with a counter of 1, turn it into a coordinate, put the edges shared with other coordinates into $\mathcal{E}_{\mathcal{C},\mathcal{C}}$ in order to adhere to Equation (4.8) to maintain compatibility, and update the counters of its neighboring fixed vertices. This is iterated until there is no fixed vertex with a counter of 1 left. If we should ever pick a fixed variable greater than 1, adding it to the coordinate set and adding the edges shared with other coordinates to $\mathcal{E}_{\mathcal{C},\mathcal{C}}$) would introduce a cycle into the tree.

Figure 4.2 illustrates this: 4.2a shows the original MRF and 4.2b shows an intermediate algorithm state with a tree of six coordinates. In 4.2c, a vertex (blue) with a previous counter of 1 was randomly selected and added to the tree, its edge to another coordinate was turned into a link (blue arrow), its edges to fixed variables were turned into dependencies (solid gray), and the counters of its fixed neighbors were incremented. In the next step, 4.2d, the last remaining vertex with a counter of 1 was added and its neighbors and edges to neighbors updated. In Figure 4.2e we demonstrate what would happen if a vertex with a counter >1 was included in the

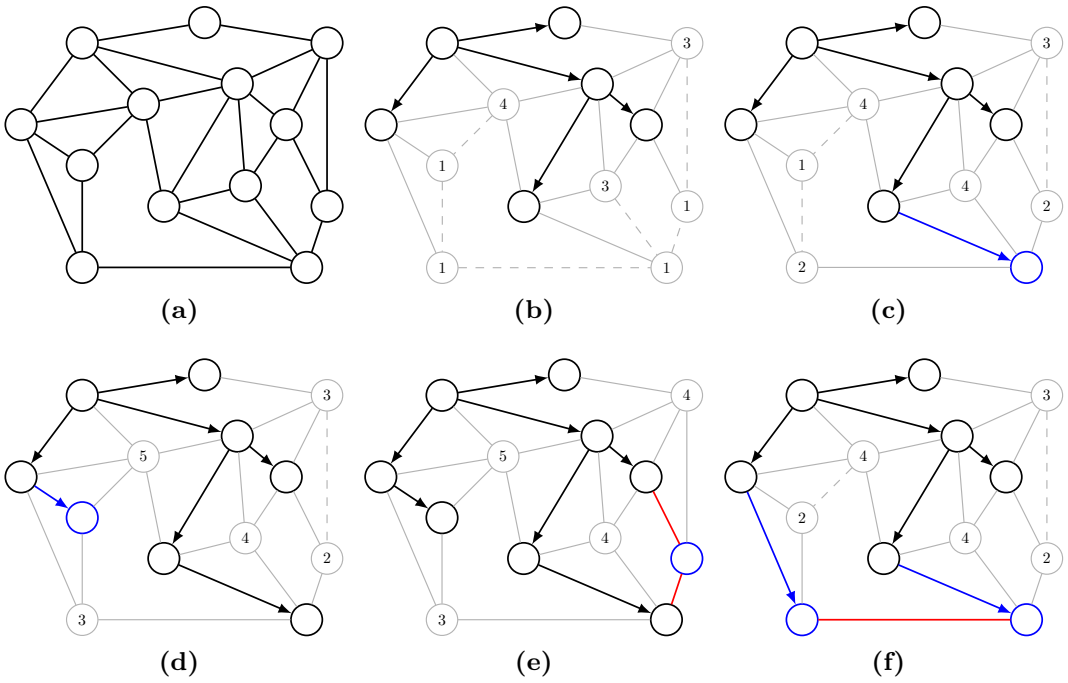


Figure 4.2.: Growing a compatible coordinate tree: (a) Original MRF taken from the paper [Thuerck et al. 2016]. (b) Coordinate tree (black: coordinates and links; solid gray: fixed variables and dependencies; dashed gray: edges between fixed variables). (c) A vertex (blue) being added to the tree. (d) One more vertex (blue) being added to the tree. (e) A vertex with a counter > 1 being added to the tree closes a cycle (red). (f) Two vertices (blue) being added to the tree of (b). Each could be added on its own, but if two threads add both in parallel, they close a cycle (red).

coordinate tree: To adhere to Equation (4.8) (maintaining compatibility) we must turn the two red edges into links, which closes a cycle. Therefore, growing must be stopped when no vertex with a counter of 1 is left. To be able to quickly select random vertices with a counter ≤ 1 during growing and quickly decide when growing must be terminated, we keep a list of all vertices with a counter ≤ 1.

Parallelization: Parallelizing this algorithm on uniform memory access machines (*e.g.*, a multi-core CPU or a GPU) works as follows: In contrast to a sequential implementation, a thread trying to add a vertex to the tree has to lock it and its in-tree neighbor before doing so. Further, counter increments have to be done atomically.

In addition to this, parallel execution has another major pitfall: Figure 4.2f shows a configuration where two vertices (blue) are tree inclusion candidates and while each of them could be added by itself, adding both of them in parallel closes a cycle (red). This case could be prevented with more extensive locking but this produces a lot of overhead and is overly cautious on problems with few threads (*e.g.*, 32 threads

from a 32-core CPU) on huge MRFs ($> 10^7$ vertices) where threads barely ever interfere with each other in such a way. The less expensive way is to let this case happen, detect it, and resolve it: If a vertex newly included in the tree has two or more coordinate neighbors, a conflict occurred and we roll back all but one of the inclusions involved in this conflict. We can, *e.g.*, roll back all involved inclusions except for the one with the largest thread ID.

When parallelizing the algorithm on many-core systems such as GPUs, the beginning of the tree growing can be a bottleneck because at that point the tree is not yet big enough to provide work for hundreds of threads. In this case, one can start growing from multiple roots. However, these multiple roots must also be sampled such that they obey Equation (4.8) and do not form a cycle.

4.2.2. Heuristics

To speed up convergence of the MRF solver, mapMAP employs two heuristics that theoretically lose compatibility with the original MRF but work very well in practice.

4.2.2.1. Spanning Trees

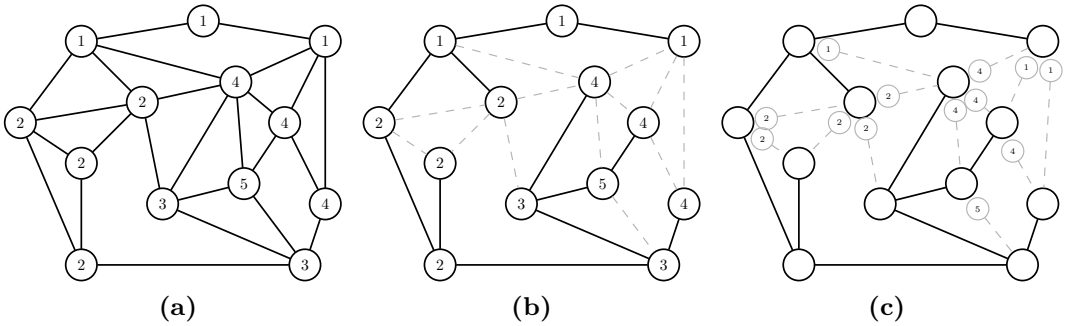


Figure 4.3.: mapMAP's spanning tree heuristic: (a) Original MRF with labels from a previous solver iteration. (b) A spanning tree (black) grown on the same MRF. Some edges (dashed) are not included in the tree. (c) Spanning tree (black) with vertex copies (gray) and dependencies (dashed gray). For each copy+dependency pair there should actually be another pair in the opposite direction, but we only show one of both to avoid cluttering the illustration.

The first heuristic is based on the spanning trees of Veksler [2005]. As mentioned earlier, Veksler's trees cannot be used in an iterative solver because they cannot "remember" the solution of a previous iteration.

We suggest a solution that enables them to do so: Figure 4.3b shows a spanning tree based on the MRF in 4.3a. For each of the (dashed) left out edges $\{v_i, v_j\}$ we then do the following: We create copies of v_i and v_j called $v_{i,\text{copy}}$ and $v_{j,\text{copy}}$, assign to them the labels of v_i and v_j, respectively, from the previous solver iteration and

create dependencies $\{v_i, v_{j,\text{copy}}\}$ and $\{v_j, v_{i,\text{copy}}\}$. These copies and copy dependencies are illustrated in Figure 4.3c although for every dropped edge we only show one copy and dependency instead of two to avoid cluttering the illustration. The copies give the whole MRF a "memory" of the labels from the previous iteration and the copy dependencies allow the solver to respect this "memory".

Parallelization: Spanning trees can be grown more easily than the compatible coordinate trees above. We do not need to keep counters or detect and roll back conflicts. A thread simply selects a random tree vertex with non-tree neighbors (again we can keep a list of tree inclusion candidates to speed up the random selection), locks it, selects one of its non-tree neighbors, locks it as well and adds it to the tree. Further we keep track of the tree's size and terminate when the size reaches $|\mathcal{V}|$.

4.2.2.2. Region Graphs

As discussed earlier, many computer vision problems are piecewise smooth or constant. Therefore many solvers group vertices into super-vertices[54] and solve the resulting graph of super-vertices, which is considerably smaller than the original. This assumes that grouped vertices should "act" in a consistent way, *i.e.*, jointly keep or change their labels in the next solver iteration. Variable grouping can, *e.g.*, be done based on the MRF's structure [Felzenszwalb and Huttenlocher 2006, Bagon and Galun 2012] or the smoothness cost function at hand [Kim et al. 2011].

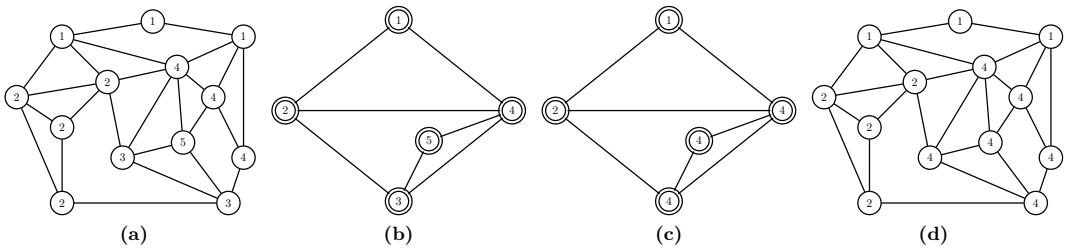


Figure 4.4.: mapMAP's region graph heuristic: (a) Original MRF with labels from a previous solver iteration. (b) Region graph with super-vertices that correspond to vertices sharing a label. (c) The same region graph after optimizing it with another algorithm. (d) The original MRF after propagating the region graph's labels back into the MRF.

We group them based on their labels from a previous solver iteration: Figure 4.4a shows an MRF with labels from a previous solver iteration. In 4.4b we grouped vertices with identical labels into super-vertices.[55] We call this graph of super-vertices a *region graph* and we solve it with one iteration of either our compatible coordinate trees or our spanning trees.

[54] also called "segments", "regions", "super-pixels", etc.

[55] In this particular example we focus on the case of a piecewise constant problem such as our texturing that uses the Potts model as smoothness term.

Our region graphs are not guaranteed to decrease the original MRF's energy. Kim et al. [2011] call an MRF's partitioning into regions *label-consistent* if they are identical with how one would partition the original MRF's globally optimal MAP solution. According to them, "if the [MRF] partitioning is indeed label-consistent, then the energy minimizing solutions under both the original and scale-reduced (based on fewer variables) energies are the same, and thus minimization of the scale-reduced energy will lead to the MAP solution of the original problem." This means that to obtain a perfect region graph, *i.e.*, one that has the same solution as the original MRF, we need to know the regions of the original MRF's solution. This seems to be not much simpler than directly solving the original MRF and without such a perfect region graph we lose compatibility. In practical problems, however, our region graphs are a good approximation and successful in fusing smaller regions into larger regions on piecewise smooth/constant problems.

Parallelization: The parallel construction of region graphs is different for multi-core systems (CPU) and for many-core systems (GPU): For multi-core systems we first search for all edges that connect vertices to be fused. We then put each vertex in its own linked list, start a multi-start breadth-first search, and merge the linked lists. For GPUs – where it is considerably harder to operate on flexible data structures *efficiently* – we instead assign a unique ID to each vertex, iterate over all edges in parallel and when we find an edge with two vertices to be fused, we assign the smaller of both vertices' IDs to both vertices. We repeat this process until no further vertex ID changes take place. Then, all vertices with a common ID are converted into one super-vertex and edges between vertices with different IDs are converted into super-edges.

Further, super-vertices and super-edges must store some information about the vertices/edges they are comprised of in order to later compute the MRF's objective function correctly. In simple cases (data and smoothness cost functions that are not spatially varying) it is sufficient to remember the multiplicity of the super-vertices and super-edges, *i.e.*, how many original vertices/edges they are comprised of.

4.2.3. Solving Trees with Dynamic Programming

All of the above steps ultimately yield trees. We solve them similarly to Veksler [2005] and Szeliski [2010, Section B.5.2] but need to add smoothness terms from dependencies into the objective function. These behave like the smoothness terms of normal edges, but the label of one of the involved vertices is fixed.

Parallelization: In the dynamic programming procedure, the computations for a vertex only depend on its children, grandchildren etc. Therefore, for a pair of vertices which are not each other's ancestors, the computation is independent, allowing parallelization over independent vertices. We can, *e.g.*, compute all children of a vertex in parallel since the children are not ancestors of each other. Further, the

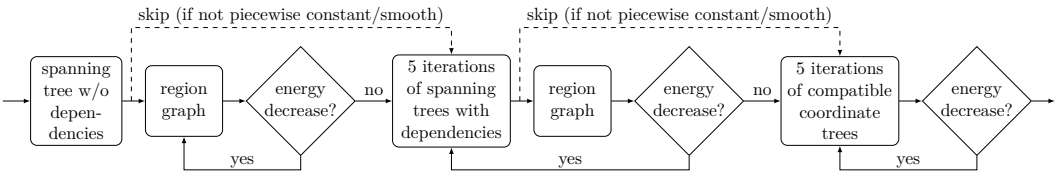


Figure 4.5.: The overall workflow of mapMAP.

computations for one vertex can be parallelized over all labels in $\mathcal{L}$. The first parallelization scheme is coarse-grained and the second is fine-grained, which yields enough parallelism to keep even GPUs busy: We assign independent vertices to different CUDA warps and labels to threads within a warp. Consequently, mapMAP is only efficient if the MRF to be solved has a large topology and label set. Otherwise, we cannot keep all warps or all threads within a warp occupied.

4.2.4. Putting the Parts Together

Now that we have introduced all building blocks of mapMAP, we will explain how they are put together (see Figure 4.5 for a flow graph). To quickly initialize the MRF with a good solution, we run one iteration of our spanning trees. Since all vertices have no prior labels, this has to be done without vertex copies and dependencies. Therefore, the first iteration is equivalent to Veksler's [2005] spanning trees. We then successively fuse regions with region graphs until we cease to improve the energy. As already mentioned, the region graphs only yield a perfect solution if the regions of the region graph before optimization happen to be identical with the regions of the MRF's optimal solution. Since this is likely not the case, we need to escape this local minimum. We therefore perform five iterations of spanning trees with dependencies that potentially move vertices between regions, split regions etc., followed by one region graph iteration to merge the resulting regions again. This procedure is iterated until a full 5+1 cycle fails to decrease the energy. After this, we optimize using our compatible coordinate trees – which are guaranteed not to increase the energy – and repeat that until a cycle of five successive iterations of compatible coordinate trees cease to decrease the energy.

4.3. Evaluation of mapMAP on Texturing Datasets

As already discussed, mapMAP is not restricted to, *e.g.*, grid-shaped topologies such as Chen and Koltun [2014] but handles MRFs with arbitrary structures. Further, it handles very large MRFs with tens of millions of vertices and hundreds of labels and scales well to CPUs with many cores and even to GPUs. These were design criteria in order to efficiently solve the view selection of our texturing algorithm (Section 3.3.2). In addition, it handles label costs and arbitrary (*e.g.*, non-submodular or non-metric) smoothness cost types because the dynamic programming with which we solve the

trees has no restrictions on the cost type. These are not requirements for our texturing application because it has submodular smoothness costs (the Potts model) and no label costs. The solver's paper [Thuerck et al. 2016] shows and discusses various other computer vision and computer graphics applications (*e.g.*, stereo, mesh segmentation, or graph coloring), some of which do use these features. In the following, we will not look at these, but only analyze the solver's runtime behavior on several differently sized texturing datasets.

4.3.1. Runtimes

For our analysis, we compare mapMAP with several popular MRF solvers:

- BP (Believe Propagation, *e.g.*, Weiss and Freeman [2001], Felzenszwalb and Huttenlocher [2006]. We use the implementation by Kolmogorov [2006].),
- GCO, which is Veksler's implementation of α-expansion [Boykov et al. 2001],
- DGCO [Shekhovtsov and Hlavác 2013], which is α-expansion with the internal graph cut algorithm replaced with a distributed graph cut variant,
- FastPD [Komodakis and Tziritas 2007], and
- TRW-S [Kolmogorov 2006].

We do not evaluate parallel BP or TRW-S variants because Kolmogorov [2006] notes that their convergence behavior is inferior to sequential BP/TRW-S. Some other solvers such as the GCO variant of Jamriska et al. [2012] or the BCD scheme of Chen and Koltun [2014] are infeasible on our texturing MRFs because they are restricted to regular topologies.

For testing we use six differently sized datasets, namely the Citywall-20, Citywall-40, Citywall-100, Reader-20, Reader-40, and Reader-100 datasets. Table 3.1 lists their properties. They consist of 0.7–20 million vertices and more than 500 labels. We mentioned earlier, that only 10–14 % of these labels are on average feasible for each vertex, which is the reason why α-expansion is inefficient on them.

As test systems we use a dual Xeon E5-2650 (256 GB RAM and 2×8 cores) and a GeForce Titan X (12 GB VRAM and 3072 CUDA cores).

The results can be found in Figure 4.6. On all datasets mapMAP finds a solution less than 2 % away from the best solution *after at most 20 seconds.* GCO is the strongest competitor: Its final energy is mostly on par with mapMAP's but it typically takes 1–2 orders of magnitude longer. On the smaller datasets Citywall-20, Citywall-40, and Reader-20 its final energy is even slightly better than mapMAP's. BP, TRW-S, and FastPD are somewhat slower than GCO on the small datasets, BP and FastPD cannot optimize the ...-40 and ...-100 datasets, and TRW-S cannot optimize the ...-100 datasets without exceeding our system's memory capabilities. DGCO (*i.e.*, parallel GCO) performs much worse than GCO in all cases.

mapMAP really excels on large datasets: On Citywall-100 and Reader-100 its energies after about 30 seconds are as good as GCO's after about 2,000 seconds. We note that mapMAP is not that superior on each and every dataset that we evaluated in the paper [Thuerck et al. 2016]. Especially on relatively small datasets, such as

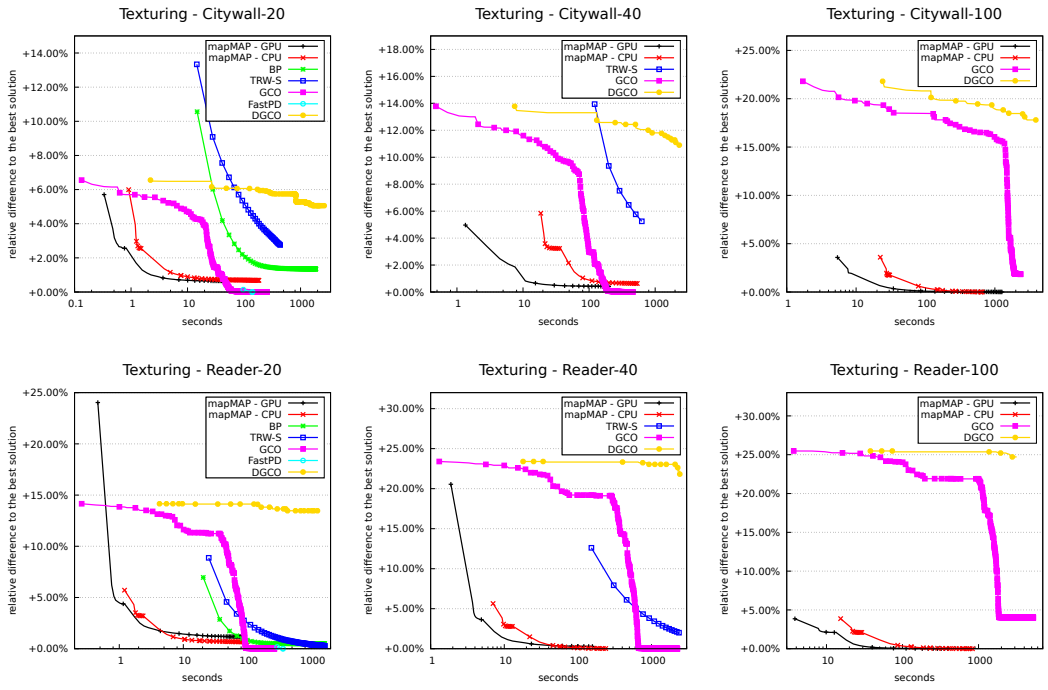


Figure 4.6.: Energy over time for two mapMAP variants (CPU and GPU) and five other MRF solvers on six differently sized texturing datasets (Table 3.1 lists their size. Note, that the x-axis (time in seconds) is logarithmic. The y-axis is the relative energy difference compared to the final solution found by the best solver. Wherever a solver is missing from the plot, its memory requirements were too heavy for these datasets, even for our relatively well-equipped machines.

the Teddy stereo problem [Szeliski et al. 2008] or the Brain-9mm dataset [Kappes et al. 2015], mapMAP typically takes longer than the best competitors to reach the same energies. However, we can summarize that mapMAP

1. is faster than all competitors by orders of magnitude on very large datasets, which is (arguably) a more interesting case than small MRFs,
2. is flexible with respect to the MRF's structure in contrast to some other solvers which require regular structures such as grids,
3. is flexible with respect to the type of smoothness costs in contrast to some other solvers that require, *e.g.*, submodular functions,
4. (not shown in this chapter but in the paper:) supports label costs which is currently only true for very few solvers.

4.3.2. Influence of the Heuristics on Convergence

The fact that the view selection's smoothness term prefers solutions with large, piecewise constant regions raises the question of how much of the energy improvement can be attributed to mapMAP's region graph heuristic and the other heuristics.

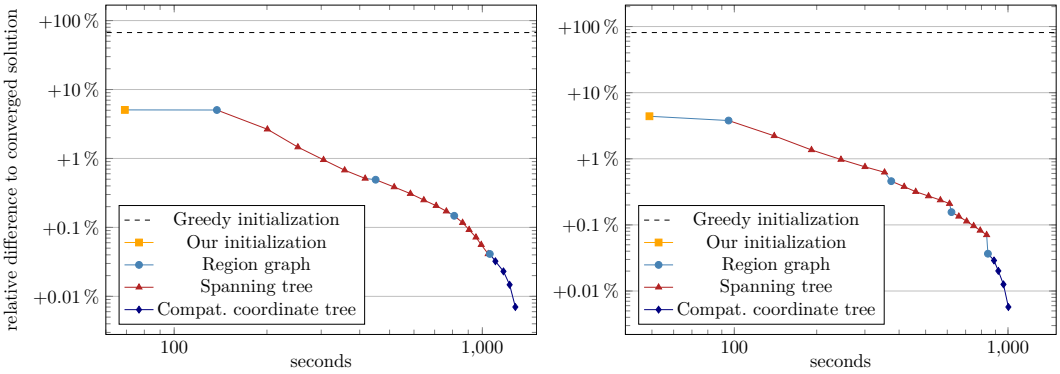


Figure 4.7.: Energy over time (both in logarithmic scale) for mapMAP's CPU implementation on Reader-100 (*left*) and Citywall-100 (*right*) where the energy improvements are color coded by the heuristic they were achieved with.

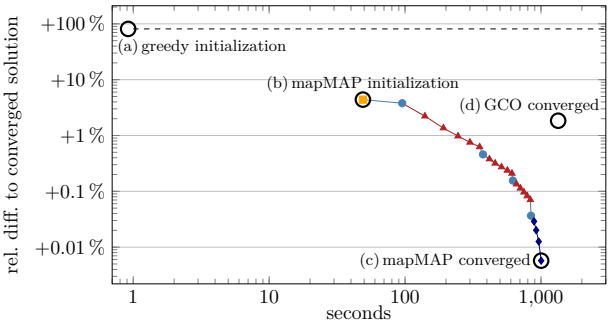
In Figure 4.7 we took our two largest datasets and color coded the energy improvements by which heuristic they were achieved with. A naïve initialization that greedily minimizes the MRF's data term is indicated with a dashed line. It is 67 % worse than the converged solution for the Reader-100 and 81 % worse for the Citywall-100. Most of the energy improvement is achieved within about a minute by our initialization with spanning trees without dependencies and the following steps only improve the energy by about 5 %.[56] At least on Reader-100 the region graph heuristic performs much weaker than we anticipated.[57]

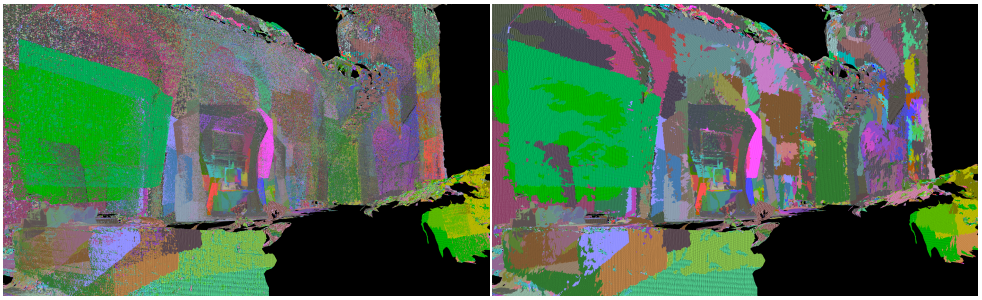
In Figure 4.8 we show the energy-over-time graph for Citywall-100 again, but with additional circles highlighting four data points: (a) a greedy initialization where we optimized with respect to all nodes' data terms only, (b) mapMAP's initialization, (c) mapMAP's converged result, and (d) GCO's converged result. The corresponding labelings for these three data points are shown in Figures 4.8(a)–(d). Figures 4.8(e) and (f) additionally show the final textured results corresponding to (c) and (d).

The greedy initialization's labeling (a) already hints at mapMAP's and GCO's converged results (c) and (d) but with a huge number of small label islands. The mapMAP initialization (b) is relatively close to mapMAP's converged result (c) but in certain areas it is still a bit rough and in practice one may not want to stop there and run the solver a bit longer. Interestingly, GCO's converged result (d) is a bit smoother than mapMAP's (c) and contains less islands despite its energy being almost 2 % worse. mapMAP's and GCO's final textured results (e) and (f) are virtually indistinguishable even under close inspection, suggesting that it may be acceptable to tweak mapMAP's stopping criterion such that it stops a bit earlier – before returns diminish drastically.

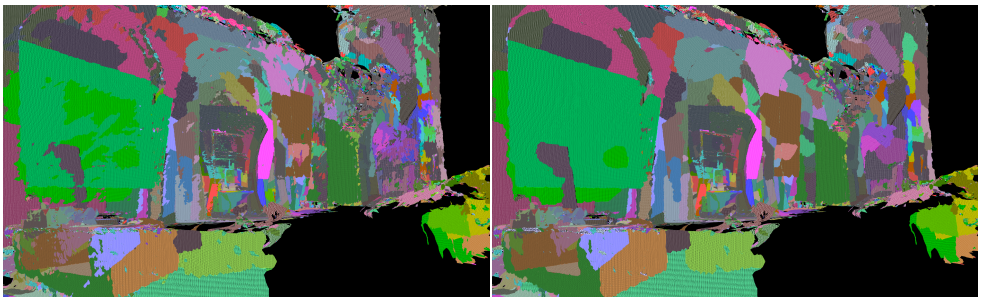
[56] Note that this is not generally the case. On, *e.g.*, Reader-20 and Reader-40 in Figure 4.6 the initialization is not very good and the following steps are indispensable to obtain a good result.

[57] Again note that this is problem dependent. Region graphs work better on Citywall-100 and in the paper [Thuerck et al. 2016, Figures 6(a) and 6(b)] we show that they work well on plane sweeping and not so well on the stereo Teddy [Scharstein and Szeliski 2003, Szeliski et al. 2008].





(a) Labeling from greedy initialization **(b)** Labeling from mapMAP's initialization



(c) Labeling from mapMAP **(d)** Labeling from GCO

(e) Textured result from mapMAP **(f)** Textured result from GCO

Figure 4.8.: Labeling visualization (colors represent input views chosen by the MRF optimization) for (a) a greedy MRF initialization, (b) mapMAP's initialization, (c) mapMAP's converged result, (d) GCO's converged result, and textured results for (e) mapMAP's converged result and (f) GCO's converged result.

We now conclude this part of the thesis: In Chapter 3 we presented the first comprehensive texturing framework for large real-world 3D reconstructions. Typical effects such as camera parameter and geometry inaccuracies, lighting and exposure variation, image scale variation, out-of-focus blur, and unreconstructed occluders are handled automatically and efficiently. The resulting high-quality textures significantly increase the realism of reconstructed models. In the context of image-based modeling and rendering (IBMR), such a framework closes the gap from an IBM system to a full IBMR system: While reconstructed models can be rendered with vertex colors [Frahm et al. 2010], a good texture allows rendering photo-realistic results, thereby enabling good predictions of novel points on the plenoptic function (Equation (1.2)).

While a view-dependent texture or unstructured Lumigraph rendering may be advantageous and desirable in certain settings, assigning a high-quality *static texture to static geometry* is nevertheless a fundamentally important building block because the resulting scene representations can easily be integrated into most existing rendering systems such as video games. This is also underlined by the fact that the code for our texturing project has in the meantime been used in various other works inside and outside of computer vision [Tscharf et al. 2015, Hernández et al. 2016a,b, Handa et al. 2016, Hafeez et al. 2016, Rothermel 2016, Holzmann et al. 2016, Li et al. 2016, Djurdjani and Laksono 2016, Utomo and Wibowo 2017, Rumpler et al. 2017, Roberts et al. 2017, Lin et al. 2017, Lurie et al. 2017], and in the EU Seventh Framework Programme research projects Harvest4D and CRPlay. Further, a commercial photogrammetry solution for the robust reconstruction of city-scale datasets, SURE by nFrames, uses a variation of our approach in its texturing implementation.

In Chapter 4 we introduced our Markov random field solver mapMAP which is an excellent solver for our texturing problems due to its support of arbitrary MRF structures and remarkable speed on datasets with $> 10^7$ vertices and hundreds of labels. Especially on our two larges datasets, the initialization with spanning trees contributed to this to a great extent.

One avenue for future work would be to parallelize the view selection optimization on a coarser scale by splitting up the mesh into sub-meshes whose optimization can then be offloaded to different nodes of a computation cluster. After all parts are finished, the seams between sub-meshes would have to be post-processed to make them unnoticeable, but this could be done with the main algorithm's building blocks (view selection and luminance adjustment, but with boundary conditions), and it could again be offloaded to cluster nodes. This way no node would require communication with the other nodes except for receiving its part of the scene and returning its result. Such a coarse parallelization scheme would be relatively easy to implement and definitely be necessary in order to process, *e.g.*, city-scale datasets.

Part II.

Evaluating Image-Based Modeling and Rendering Systems

We now come to the part that is eponymous for this thesis. Given a 3D reconstruction system such as our texturing system described in Chapter 3, the question that arises is: "How good is our IBMR system/model?" Many evaluation metrics and benchmarks focus on the geometric accuracy of reconstructions. However, the ultimate goal of many systems is not accurate geometric reconstruction but rather the rendering of photo-realistic novel views of a scene without visible artifacts. In that context, geometric accuracy is a poor predictor of visual accuracy as we shall see later. Furthermore, evaluating only geometric accuracy by itself does not allow evaluating systems that lack a global, geometric scene representation (examples include a light field or view-dependent geometry) and is misleading for IBR systems (*e.g.*, unstructured Lumigraphs) that work with coarse proxy geometry (which may be inaccurate geometrically but may do a good job in image-based rendering).

We propose a unified evaluation approach based on novel view prediction error that is able to analyze the visual quality of *any* image-based modeling and rendering system, because its only requirement is that systems can render novel views from input images. A key advantage of this approach is that it does not require ground-truth geometry, which dramatically simplifies the creation of test datasets and benchmarks. It also allows us to evaluate the quality of unknown scenes during the acquisition and reconstruction process, which is useful for acquisition planning.

In Chapter 5 we will give a motivation for why a geometric evaluation alone is not sufficient for evaluating image-based modeling and rendering systems and we give a review of the literature on image-based modeling and rendering evaluation. We then describe our evaluation methodology in Chapter 6, before coming to the main part – Chapter 7 – where we evaluate whether our methodology fulfills some basic requirements, how it relates to existing geometry-based benchmarks, and how it correlates with human judgment on visual reconstruction quality. In Chapter 8 we demonstrate an important application of our approach – the localization of errors in IBR systems with a global surface mesh – and present a new benchmark for image-based modeling and rendering systems based on our approach.

Finally, in Chapter 9 we conclude this thesis by summarizing its contributions and pointing out avenues for future work.

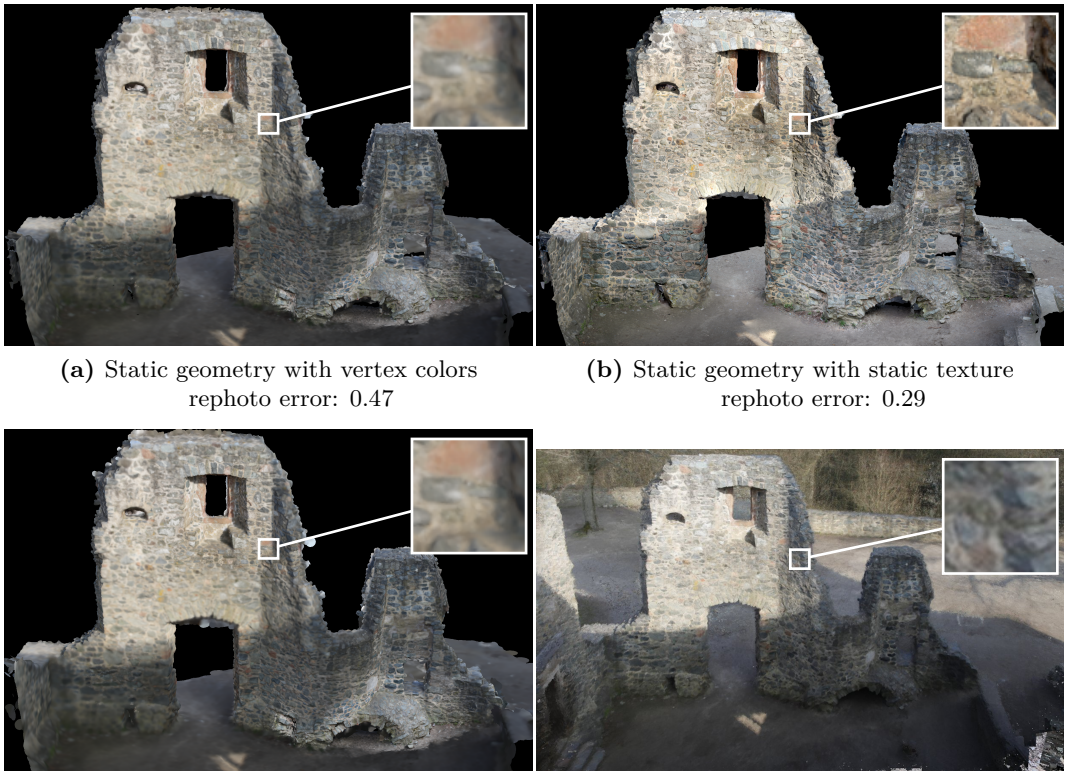
5. Introduction to IBMR Evaluation

Intense research in the computer vision and computer graphics communities has lead to a wealth of image-based modeling and rendering systems that take images as input, construct a model of the scene, and then create photo-realistic renderings for novel viewpoints. The computer vision community contributed tools such as structure from motion and multi-view stereo to acquire geometric models that can subsequently be textured. The computer graphics community proposed various geometry- and image-based rendering systems. Some of these, such as the unstructured Lumigraph [Buehler et al. 2001], synthesize novel views directly from the input images (and a rough geometry approximation), producing photo-realistic results without relying on a detailed geometric model of the scene. Even though remarkable progress has been made in the area of modeling and rendering of real scenes, a wide range of issues remain, especially when dealing with complex datasets under uncontrolled conditions. In order to measure and track the progress of this ongoing research, it is essential to perform objective evaluations.

Existing evaluation efforts [Seitz et al. 2006, Strecha et al. 2008, Aanæs et al. 2016] focus on systems that acquire mesh models. They compare the reconstructed meshes with ground-truth geometry and evaluate measures such as geometric completeness and accuracy. This approach falls short in several regards: First, only scenes with available ground-truth models can be analyzed. Ground-truth models are typically not available for large-scale reconstruction projects outside the lab such as PhotoCity [Tuite et al. 2011], but there is nevertheless a need to evaluate reconstruction quality and identify problematic scene parts. Second, evaluating representations other than meshes is problematic: Point clouds (see Figure 5.1c) can only partially be evaluated,[58] and image-based rendering representations (Figure 5.1d) or light fields [Levoy and Hanrahan 1996] cannot be evaluated at all. And third, it fails to measure properties that are complementary to geometric accuracy. While there are applications for which only geometric accuracy matters (*e.g.*, reverse engineering or 3D printing), applications that produce renderings for human consumption are arguably more concerned with visual quality. This is for instance the main focus in the unstructured Lumigraph [Buehler et al. 2001], where the geometric proxy does not necessarily have to be accurate and only visual accuracy of the resulting renderings matters.

If we consider, *e.g.*, 3D reconstruction pipelines, for which geometric evaluations such as the common multi-view benchmarks [Seitz et al. 2006, Strecha et al. 2008,

[58] At least according to the Middlebury [Seitz et al. 2006] definition, reconstruction accuracy can be measured but completeness cannot, because that requires drawing perpendiculars from ground-truth mesh vertices onto a reconstructed surface, which a point cloud does not have.



(a) Static geometry with vertex colors
rephoto error: 0.47

(b) Static geometry with static texture
rephoto error: 0.29

(c) Splatted point cloud
rephoto error: 0.51

(d) View-dep. geometry with view-dep. texture
rephoto error: 0.62

Figure 5.1.: Castle Ruin with different 3D reconstruction representations. Geometric evaluation methods [Seitz et al. 2006, Aanæs et al. 2016] cannot distinguish (a) from (b) as both have the same geometry. While the splatted point cloud (c) could in principle be evaluated with these methods, the IBR solution (d) cannot be evaluated at all.

Aanæs et al. 2016] were designed, we can easily see that visual accuracy is complementary to geometric accuracy. The two measures are intrinsically related since errors in the reconstructed geometry tend to be visible in renderings,[59] but they are not fully correlated and are therefore distinct measures. Evaluating the visual quality adds a new element to multi-view stereo reconstruction: recovering a good surface texture in addition to the scene geometry. Virtual scenes are only convincing if effort is put into texture acquisition, which is challenging, especially with datasets captured under uncontrolled real-world conditions with varying exposure, illumination, or foreground clutter. If this is done well, the resulting texture may even hide small geometric errors. A metric that disregards visual accuracy cannot for example tell the two reconstructions in Figures 5.1a and 5.1b apart because their geometry is identical.

[59] *E.g.*, if a wall's depth has not been correctly estimated, this may not be visible in a frontal view but definitely when looking at it at an angle.

(a) "Merrell Confidence" [Merrell et al. 2007a], geom. error: 0.83 mm, completeness: 0.88 %

(b) "Generalized-SSD" [Calakli et al. 2012], geom. error: 0.81 mm, completeness: 0.96 %

(c) "Campbell" [Campbell et al. 2008], geom. error: 0.48 mm, completeness: 0.99 %

(d) "Hongxing" [Hongxing et al. 2010], geom. error: 0.79 mm, completeness: 0.96 %

Figure 5.2.: Four submissions to the Middlebury benchmark to which we added texture (as described in Chapter 3): (a) and (b) have similar geometric error and different visual quality. Contrarily, (c) and (d) have different geometric error and similar visual quality.

A geometric reconstruction evaluation metric does not allow for directly measuring the achieved visual quality of the textured model, and it is not always a good indirect predictor for visual quality: In Figure 5.2, we textured four submissions to the Middlebury benchmark using our texturing approach (see Section 3). The textured model in Figure 5.2a is strongly fragmented while the model in 5.2b is not. Thus, the two renderings exhibit very different visual quality. Their similar geometric error, however, does not reflect this. Contrarily, Figures 5.2c and 5.2d have very different geometric errors despite similar visual quality. Close inspection shows that 5.2d has a higher geometric error because its geometry is too smooth. This is hidden by the texture, at least from the viewpoints of the renderings. In both cases, similarity of geometric error is a poor predictor for similarity of visual quality, clearly demonstrating that the purely distance-based Middlebury evaluation is by design unable to capture certain aspects of 3D reconstructions. Thus, a new methodology that evaluates visual reconstruction quality is needed.

Most 3D reconstruction pipelines are modular: Typically, structure from motion is followed by dense reconstruction, texturing, and rendering (in image-based rendering the latter two steps form one joint step). While each of these steps can be evaluated individually, our proposed approach is holistic and evaluates the complete pipeline *including* the rendering step by scoring the visual quality of the final renderings. This is more consistent with the way humans assess quality: They cannot directly assess the quality of intermediate representations (*e.g.*, a 3D mesh model), but instead look at two-dimensional projections, *i.e.*, renderings of the final model from different viewpoints.

Leclerc et al. [2000] pointed out that in the real world, inferences that people make from one viewpoint are consistent with the observations from other viewpoints. Consequently, good models of real world scenes must be self-consistent as well. We exploit this self-consistency property as follows: We divide the set of captured images into a training set and an evaluation set. We then reconstruct the training set with an image-based modeling pipeline, render novel views with the camera parameters of the evaluation images, and compare those renderings with the evaluation images using selected image difference metrics.

This treats image-based modeling and rendering algorithms entirely as a black box: Algorithms infer a model of the world based on training images and make predictions for the images from novel viewpoints. Since this is exactly the definition of image-based modeling and rendering systems (Chapter 1), our approach is the most natural evaluation scheme for them because it directly evaluates the output (*i.e.*, images) instead of internal models (*e.g.*, a surface mesh). In fact, our approach encourages that future 3D reconstruction techniques take photo-realistic renderability into account. This line of thought is also advocated by Shan et al.'s [2013] Visual Turing Test or Vanhoey et al.'s [2015] appearance-preserving mesh simplification.

Our work draws inspiration from Szeliski [1999], who proposed using intermediate frames for optical flow evaluation. We extend this into a complete evaluation paradigm that is able to handle a diverse set of image-based modeling and rendering methods. Since the idea of imitating image poses resembles computational rephotography [Bae et al. 2010] as well as the older concept of repeat photography [Webb et al. 2010], we call our technique *virtual rephotography* and call renderings *rephotos*. By enabling the evaluation of visual accuracy, virtual rephotography puts visual accuracy on a level with geometric accuracy as a quality of 3D reconstructions.

In summary, the contributions of our evaluation framework are as follows:

- A flexible evaluation paradigm using the novel view prediction error that can be applied to any renderable scene representation,
- quantitative view-based error metrics in terms of image difference and completeness for the evaluation of photo-realistic renderings,
- a thorough evaluation of our methodology on several datasets, with different reconstruction and rendering techniques, and with different image difference metrics, and
- a virtual rephotography-based benchmark.

Our system is advantageous over classical evaluation systems in that it

- allows measuring aspects complementary to geometry, such as texture quality in a multi-view stereo and texturing pipeline,
- dramatically simplifies the creation of new benchmarking datasets since it does not require a ground-truth geometry acquisition and vetting process,
- enables direct error visualization and localization on the scene representation (see Fig. 8.3), which is useful for error analysis and acquisition guidance, and
- makes reconstruction quality comparable across different scene representations

(see Figure 5.1) and thus closes a gap between computer vision (image-based modeling) and computer graphics techniques (image-based rendering).

5.1. Related Work

Image-based modeling and rendering (IBMR) is a topic that spans both computer graphics and vision. On the modeling side, the geometry-based Middlebury multi-view stereo benchmark [Seitz et al. 2006] and other factors have triggered research on different reconstruction approaches. On the image-based rendering (IBR) side, many evaluation approaches have been proposed [Schwarz and Stamminger 2009, Berger et al. 2010, Guthe et al. 2016], but most of them are too specialized to be directly transferable to other IBMR representations. The virtual rephotography framework we present here takes a wider perspective by considering complete reconstruction *and* rendering pipelines.

In the following, we first provide a general overview of evaluation techniques for image-based modeling and rendering, before discussing image comparison metrics with respect to their suitability for our proposed virtual rephotography framework.

5.1.1. Evaluating Image-Based Modeling and Rendering

Algorithms need to be objectively evaluated in order to *prove* that they advance their field [Förstner 1996]. For the special case of multi-view stereo (MVS), the first to do this was the Middlebury MVS benchmark [Seitz et al. 2006]: It evaluates algorithms by comparing reconstructed geometry with scanned ground truth, and measures accuracy (distance of the mesh vertices to the ground truth) and completeness (percentage of ground-truth vertices within a threshold of the reconstruction). The downsides of this purely geometric evaluation have been discussed on pages 81ff. In addition, the Middlebury benchmark has aged and cannot capture most aspects of recent MVS research (*e.g.*, preserving fine details when merging depth maps with drastically different scales or recovering texture).

Strecha et al. [2008] released a benchmark with larger, more realistic, architectural outdoor scenes and larger images. However, the complexity of these datasets with respect to 3D reconstruction is limited, because the scenes are still well-textured. Strecha et al. use laser scanned ground-truth geometry and compute measures similar to the Middlebury benchmark. Merrell et al. [2007b] presented a more challenging outdoor dataset of a complete building. However, there is only one dataset available and the ground truth is not measured, but hand-modeled, which does not capture all details of the underlying building even though the building is not very complex. In contrast to the other benchmarks, the authors provide video instead of photos which brings its very own set of challenges, such as rolling shutter, motion blur, huge amounts of data, *etc.* Aanæs et al. [2016] released a dataset of many controlled indoor objects with larger and higher quality images, more accurate camera positions from a pre-programmed robotic camera arm, much denser ground-truth geometry,

and with a modified evaluation protocol that is still based on geometric accuracy. The captured scenes partially exhibit complicated surface reflectance properties.

Recently, two very challenging benchmarks were published: Schöps et al. [2017] and Knapitsch et al. [2017] both released very realistic, large, complicated indoor and outdoor datasets with remarkably detailed ground truth. They include complicated illumination (bright outdoor scenes with direct sun illumination and dark indoor scenes such as a bar scene), non-Lambertian and weakly textured surfaces, and a geometric complexity that is about as complicated as it gets in the real world. Schöps et al. provide photos and camera parameters to users and Knapitsch et al. provide video data and leave the inference of camera parameters to the user.[60]

Letting users reconstruct from video data is a blessing in that it may boost research in the direction of the challenges of video data and a curse in that errors in reconstructions can no longer be pinpointed to being either reconstruction errors or a consequence of bad input data (*e.g.*, motion blur). Similarly, not providing camera parameters lets us, *e.g.*, compare the (dis)advantages of different structure from motion methods and potentially forces future researchers to do a better job in camera registration than the benchmark authors could have done. However, it also prevents precisely pinpointing whether reconstruction errors are coming from the camera registration or from the dense reconstruction. With our proposed evaluation scheme, we will later have the same issue: Widening the scope of an evaluation scheme reduces our ability to pinpoint the reconstruction step that caused an error. Knapitsch et al. [2017] partially solve this problem by, *e.g.*, running SfM algorithm SfM-A with MVS algorithm MVS-A, SfM-A with MVS-B, and SfM-B with MVS-A to check whether replacing one of the pipeline steps in-/decreases the reconstruction quality.

All of the above 3D reconstruction evaluation efforts strictly focus on geometric criteria and the objections stated against Middlebury above therefore apply to all of them. None of them addresses challenges such as photo-realistic renderability, which is somewhat orthogonal to geometric accuracy. Consequently, researchers who address non-geometric aspects still have to mostly rely on *qualitative* comparison, letting paper readers judge their results by visual inspection.

Szeliski [1999] encountered the same problem in the evaluation of optical flow and stereo and proposed novel view prediction error as a solution: Instead of measuring how well algorithms estimate flow, he measures how well the estimated flow performs in frame rate doubling. Given two video frames for time t and $t+2$, flow algorithms predict the flow between them, use the flow to predict the intermediate frame $t+1$, and this frame can then be compared with the non-public ground-truth frame. This has (among other metrics) been implemented in the Middlebury flow benchmark [Baker et al. 2011]. Leclerc et al. [2000] use a related concept for stereo evaluation: They call a stereo algorithm self-consistent if its depth hypotheses for image I_1 are the same when inferred from image pairs (I_0, I_1) and (I_1, I_2). Szeliski's (and our)

[60] Since users infer camera parameters, benchmark submissions have to be aligned to the ground-truth LiDAR scans prior to evaluation.

criterion is more flexible: It allows the internal model (a flow field for Szeliski, depth for Leclerc) to be wrong as long as the resulting rendering looks correct, a highly relevant case as demonstrated by Hofsetz et al. [2004]. Virtual rephotography is clearly related, especially to Szeliski's approach.[61] However, Szeliski only focused on view interpolation in stereo and optical flow. We extend novel view prediction error to the much more challenging general case of image-based modeling and rendering where views have to be extrapolated over much larger distances.

Novel view prediction error has previously been used in image-based modeling and rendering, *e.g.*, for evaluating the accuracy of BRDF recovery [Yu et al. 1999, Section 7.2.3]. However, Yu et al. only show qualitative comparisons to their readers and do not reason about the properties or usefulness of this measure. The same holds for the Visual Turing Test: Shan et al. [2013] ask study participants to compare photos and novel view renderings at varying resolutions to obtain a *qualitative* judgment of realism. Similarly, albeit on much simpler datasets, Weigel and Fan [2008] let study participants judge the quality of novel views (in a trifocal tensor based transfer method, *cf.* "rendering with implicit geometry" on page 30f), but they do not show reference images to the participants.

Fitzgibbon et al. [2005, Figure 7d] and Utomo and Wibowo [2017, Table 3] use novel view prediction error to quantify the error of their IBR method and 3D reconstructions respectively, but they neither reflect on the novel view prediction error's properties nor compare their results with those of other IBR/reconstruction methods. Further, Fitzgibbon et al.'s image difference metric, RGB difference, only works in scenarios where all images have identical camera response curves, illumination, exposure, *etc.* We show later that a very similar metric ($\Delta C_b + \Delta C_r$ difference in $\mathrm{YC_bC_r}$ color space) fails in settings that are more general than Fitzgibbon et al.'s.

Mueller et al. [2005] attempt a reflection on the properties of novel view prediction error but their evaluation falls short in that they only present three extremely simple experiments with a handful of data points that are not sufficient to build trust in the usefulness of this method.

Morvan and O'Sullivan [2009] use novel view prediction error for choosing views that, if removed from an unstructured Lumigraph, degrade the result quality the least. In a user study with relatively large and realistic datasets, they check how many views they can remove (in a greedy fashion) until users notice, and which image difference metric performs best in this task. However, they do not directly correlate user ratings and computer generated quality scores and their setup of greedily removing views and checking when users detect the degradation has some uncertain elements: Detectable degradation for example cannot only result from a bad image difference metric but also from an unfortunate (*i.e.*, only locally optimal) choice in the greedy removal algorithm.

In contrast to Shan et al. [2013] or Weigel and Fan [2008], we propose to automate the evaluation process by comparing renderings and original images from the evaluation set using several image difference metrics to *quantitatively* measure

[61] For this reason the title of this thesis was chosen to be very similar to Szeliski's [1999] title.

and localize reconstruction defects. We suggest the novel view prediction error as a general, quantitative evaluation method for the whole field of image-based modeling and rendering, and – in the spirit of Pont-Tuset and Marques [2013, 2016] (see Section 5.1.3) and in contrast to all of the above papers – present a comprehensive evaluation to shed light on its usefulness for this purpose.

In image-based rendering, a variety of works cover the detection of somewhat specialized artifacts: Schwarz and Stamminger [2009], Berger et al. [2010], and Guthe et al. [2016] automatically detect ghosting and popping artifacts in IBR. Vangorp et al. [2011] investigate how users rate the severity of ghosting, blurring, popping, and parallax distortion artifacts in street-level IBR subject to parameters such as scene coverage by the input images, number of images used to blend output pixels, or viewing angle. In later work, Vangorp et al. [2013] evaluate how users perceive parallax distortions in street-level IBR (more specifically distortions of rectangular protrusions in synthetic façade scenes rendered with planar IBR proxies) and derive metrics that indicate where in an IBR scene a user should be allowed to move or where additional images of the scene should be captured to minimize distortions. Their final metric is independent of the actual visual appearance of the rendered scene and only takes its geometry into account. In video-based rendering, Tompkin et al. [2013] analyze user preferences for different types of transitions between geometrically connected videos and how users perceive the artifacts that can occur. None of the above papers' results can be easily transferred to non-IBR scene representations such as textured MVS reconstructions and some of them cannot even be easily generalized to IBR of general scenes.

Pagés et al. [2011] compare the quality of different texture atlases that correspond to the same mesh, which is, *e.g.*, useful for comparing texture compression techniques. But like the IBR artifact detectors above, this metric is specialized for one scene representation type (here: static geometry with static texture).

A metric that evaluates the quality of polygonal 3D reconstruction and, similarly to ours, does not require geometric ground-truth data, is Hoppe et al.'s [2012a] metric for view planning in multi-view stereo. For a mesh model, it checks each polygon's degree of visibility redundancy and maximal resolution. In contrast to our method, it does not measure visual reconstruction quality itself, but it measures circumstances that are assumed to cause reconstruction errors.

5.1.2. Image Comparison Metrics

Our approach reduces the task of evaluating 3D reconstructions to comparing pairs of images, namely evaluation images with their rendered counterparts. We already went through some image difference metrics in the context of stereo matching costs in Section 2.2.2.2, which is why we will cover the topic at a higher level here.

In Section 2.2.2.2, we learned that image comparison metrics need to be somewhat invariant under typical image transformations such as low-frequency luminance or contrast changes and therefore need to operate on a neighborhood region around a pixel. Examples include the structural similarity index (SSIM) [Wang et al. 2004],

Figure 5.3.: *Left to right*: Input image, rendering from the same viewpoint, and HDR-VDP-2 result (color scale: VDP error detection probability)

normalized cross-correlation (NCC), Census [Zabih and Woodfill 1994], and zero-mean sum of squared differences (ZSSD).

From a conceptual point of view, image comparison in a photo-realistic rendering context should be performed with human perception in mind. The visual difference predictor VDP [Daly 1993] and its high dynamic range variant HDR-VDP-2 [Mantiuk et al. 2011] are based on the human visual system and detect differences near the visibility threshold. Since reconstruction defects are typically significantly above this threshold, these metrics are too sensitive and unsuitable for our purpose. As Figure 5.3 shows, HDR-VDP-2 marks almost the whole rendering as defective compared to the input image. For HDR imaging and tone mapping Aydın et al. [2008] introduced the dynamic range independent metric DRIM, which is invariant under exposure changes. However, in our experience DRIM's as well as HDR-VDP-2's range of values is hard to interpret in the context of reconstruction evaluation. Finally, the visual equivalence predictor [Ramanarayanan et al. 2007] measures whether two images have the same high-level appearance even in the presence of structural differences. However, knowledge about scene geometry and materials is required, which we explicitly want to avoid. Given these limitations of perceptually-based methods, we utilize more basic image correlation methods. Our experiments show that they work well in our context.

5.1.3. Meta-Evaluation

A large fraction of our experiments (Chapter 7) will be devoted to meta-evaluating our evaluation scheme in combination with various image comparison metrics. Pont-Tuset and Marques [2013, 2016] presented a similar work in a different field: They meta-evaluated different evaluation metrics for image segmentation. They were confronted with the fact that there is a wealth of metrics in image segmentation and it is not immediately clear what makes for a good metric.

Pont-Tuset and Marques solved this by coming up with a series of very basic assumptions: They assume that any reasonable image segmentation metric should rank a state-of-the-art segmentation algorithm's segmentations higher than "fake" segmentations, *i.e.*, segmentations that ignore the actual image content. More specif-

ically, they assume that, given an image A, its ground-truth segmentation A_{gt} (*e.g.*, from the Berkeley segmentation dataset [Martin et al. 2001]), a state-of-the-art algorithm's segmentation $\mathrm{seg}(A)$, and an error metric $e(A_{\mathrm{gt}}, \mathrm{seg}(A))$, that

- $e(A_{\mathrm{gt}}, \mathrm{seg}(A)) < e(A_{\mathrm{gt}}, \mathrm{seg}(B))$ for an image B that is unrelated to A,
- $e(A_{\mathrm{gt}}, \mathrm{seg}(A)) < e(B_{\mathrm{gt}}, \mathrm{seg}(A))$ for a ground-truth segmentation B_{gt} of an image B that is unrelated to A,
- $e(A_{\mathrm{gt}}, \mathrm{seg}(A)) < e(A_{\mathrm{gt}}, \mathrm{rand})$ for a random segmentation rand, and
- $e(A_{\mathrm{gt}}, \mathrm{seg}(A)) < e(A_{\mathrm{gt}}, \mathrm{grid})$ for a segmentation into a regular grid.

Pont-Tuset and Marques's assumptions are slightly circular because they require the existence of sufficiently good (*i.e.*, non-random) ground truth and segmentation algorithms, whose meta-evaluation is the exact purpose of these "axioms", but we can agree that assuming correct segmentations to be better than segmentations ignoring the image content is so very basic that it is reasonable. Surprisingly, not all metrics fulfill these conditions as often as they should (see examples where they are violated in Figures 6 and 7 of Pont-Tuset and Marques [2016]).

Even though we deal with a field that is unrelated to image segmentation, our goal is exactly the same as Pont-Tuset and Marques's: We will formulate a set of requirements that any reasonable metric should fulfill and devise a range of experiments to verify the fulfillment (maybe not on each and every image but at least statistically on many images). The power of these requirements and meta-evaluation experiments is that they allow us to distinguish between suitable and unsuitable metrics. This is in contrast to, *e.g.*, Fitzgibbon et al. [2005] who use a metric (RGB difference) without reflecting about its suitability.

6. Evaluation Methodology of Virtual Rephotography

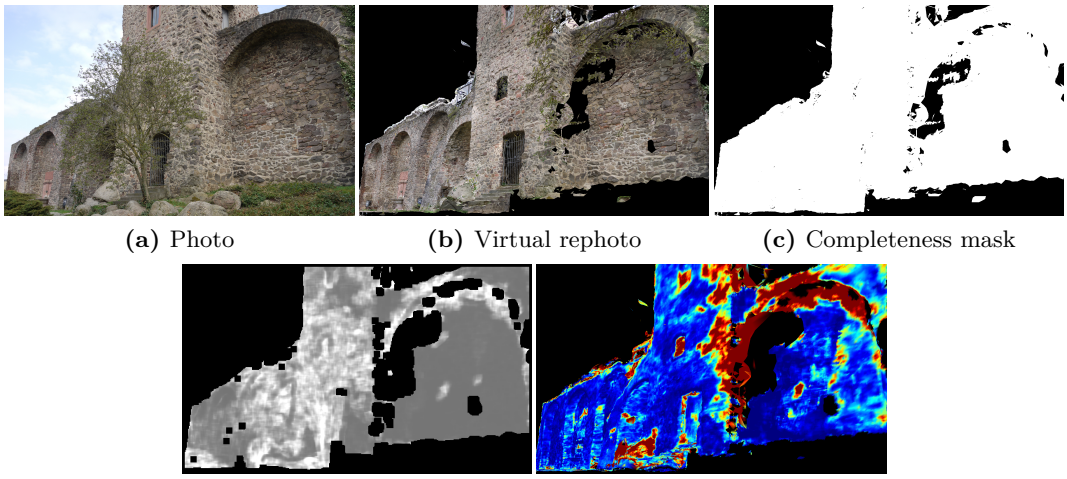
Before introducing our own evaluation methodology, we propose and discuss a set of general desiderata that should ideally be fulfilled: First, an evaluation methodology should *evaluate the actual use case* of the approach under examination. For example, if the purpose of a reconstruction is to record accurate geometric measurements, a classical geometry-based evaluation [Seitz et al. 2006, Strecha et al. 2008, Aanæs et al. 2016] is the correct methodology. In contrast, if the reconstruction is used as a proxy in an image-based rendering setting, one should instead evaluate the achieved rendering quality. This directly implies that the results of different evaluation methods will typically be inconsistent unless a reconstruction is perfect.

Second, an evaluation methodology should be able to *operate solely on the data used for the reconstruction itself* without requiring additional data that are not available per se. It should be *applicable on site* while capturing new data. Furthermore, the *creation of new benchmark datasets should be efficiently possible.* One of the key problems of existing geometry-based evaluations is that the creation of ground-truth geometry models with high enough accuracy requires a lot of effort (*e.g.*, high quality scanning techniques in a very controlled environment). Thus, it is costly and typically only used to benchmark an algorithm's behavior in a specific setting. An even more important limitation is that it cannot be applied to evaluate its performance on newly captured data.

Third, an evaluation methodology should *cover a large number of reconstruction methods.* The computer graphics community's achievements in image-based modeling and rendering are fruitful for the computer vision community and vice versa. Only being able to evaluate a limited set of reconstruction methods restricts intercomparability and exchange of ideas between the communities.

Finally, *the metric for the evaluation should be linear* (*i.e.*, if the quality of model B is right in the middle of model A and C, its score should also be right in the middle). If this is not possible, the metric should at least give reconstructions an ordering – *i.e.*, if model A is better than B, its error score should be lower than B's. In the following we will always refer to this as the *ordering criterion.* This ordering criterion is the one that we will be verifying in a similar manner to Pont-Tuset and Marques [2016]: We will devise experiments where it is intuitively clear that A should be better than B and we will verify whether our metrics indeed assign a lower error to A.

Fulfilling all of the above desiderata simultaneously and completely is challenging. In fact, the classical geometry-based evaluation methods [Seitz et al. 2006, Strecha



(a) Photo **(b)** Virtual rephoto **(c)** Completeness mask

(d) Error image (1-NCC) **(e)** Error projection

Figure 6.1.: An overview of the complete virtual rephotography pipeline.

et al. 2008, Aanæs et al. 2016] satisfy only the first and last items above (in geometry-only scenarios, it evaluates the use case and is linear). In contrast, our virtual rephotography approach fulfills all but the last requirement (it provides an ordering but is not necessarily linear).

In the following, we first describe our method, the overall workflow, and the used metrics in detail, before evaluating our method in Chapter 7 using a set of controlled experiments.

6.1. Overview and Workflow

The key idea of our proposed evaluation methodology is that the performance of each algorithm stage is measured in terms of its impact on the final rendering result. This makes the specific system to be evaluated largely interchangeable, and allows for evaluating different combinations of components end-to-end. The only requirement is that the system is able to build a model of the world from given input images and produce (photo-realistic) renderings from the viewpoints of test images.

We now give a brief overview of the complete workflow: Given a set of input images of a scene such as the one depicted in Figure 6.1a, we perform reconstruction using n-fold cross-validation. In each of the n cross-validation instances we put $1/n$ th of the images into an evaluation set E and the remaining images into a training set T. The training set T is then handed to the reconstruction algorithm that produces a 3D representation of the scene. This could, *e.g.*, be a textured mesh, a point cloud with vertex colors, or the internal representation of an image-based rendering approach such as the set of training images combined with a geometric proxy of the scene. In Chapters 7 and 8 we show multiple examples of reconstruction algorithms

that we used for evaluation.

The reconstruction algorithm then rephotographs the scene, *i.e.*, renders it photo-realistically using its own, native rendering system with the exact extrinsic and intrinsic camera parameters of the images in E (see Figure 6.1b). If desired, this step can also provide a completeness mask that marks pixels not included in the rendered reconstruction (Figure 6.1c). Note that we regard obtaining the camera parameters as part of the reconstruction algorithm. However, since the test images are disjoint from the training images, camera calibration (*e.g.*, using structure from motion [Snavely et al. 2006]) must be done beforehand to have all camera parameters in the same coordinate frame. State-of-the-art structure from motion systems are for the most part sub-pixel accurate (otherwise multi-view stereo would not work on them) and are assumed to be sufficiently accurate for the purpose of this evaluation.

Next, we compare rephotos and test images with image difference metrics and ignore those regions which the masks mark as unreconstructed (Figure 6.1d). We also compute completeness as the fraction of valid mask pixels and average completeness and error scores over all rephotos to obtain global scores for the whole dataset.

Finally, we can project the error images onto the reconstructed model to visualize local reconstruction error (Figure 6.1e).

6.2. Accuracy and Completeness

In order to evaluate the visual accuracy of rephotos, we measure their similarity to the test images using image difference metrics. The simplest choice would be the pixel-wise mean squared error. The obvious drawback is that it is not invariant under luminance changes. If we declared differences in luminance as a reconstruction error, we would effectively require all image-based reconstruction and rendering algorithms to, *e.g.*, produce illumination effects during rendering. However, only a few reconstruction algorithms currently recover the true albedo and reflection properties of surfaces as well as the scene lighting (examples include the works by Haber et al. [2009] and Shan et al. [2013]). An evaluation metric that only works for such methods would have a very small scope. Furthermore, in real-world datasets, illumination can vary among the input images and capturing the ground-truth illumination for the test images would drastically complicate our approach. Thus, it seems adequate to use luminance-invariant image difference metrics.

We therefore use the $\mathrm{YC}_b\mathrm{C}_r$ color space and sum up the absolute errors in the two chrominance channels. We call this error $\Delta C_b + \Delta C_r$ *error* in the following. However, this metric takes only single pixels into consideration and detects in practice mostly minor color noise.

Thus, we also analyze patch-based metrics. Some of these metrics were already introduced in Section 2.2.2.2 because they are well-established tools in computer vision (*e.g.*, in stereo or optical flow) precisely for our use case – comparing image similarity in the presence of changes in illumination, exposure etc. In particular, we analyze

- zero-mean sum of squared differences (ZSSD),
- Wang et al.'s [2004] structural dissimilarity index $\text{DSSIM} = {}^{(1-\text{SSIM})}/_{2}$,
- normalized cross-correlation (we use 1-NCC instead of NCC to obtain a *dis*similarity metric),
- Census [Zabih and Woodfill 1994], and
- six variants of Preiss et al.'s [2014] improved color image difference iCID: iCID perceptual, hue-preserving, saturating, perceptual CSF, hue-preserving CSF, and saturating CSF.

In conjunction with the above accuracy measures, one must always compute some completeness measure that states the fraction of the test set for which the algorithm made a prediction. Otherwise algorithms could resort to rendering only the scene parts for which they are certain about their prediction's correctness. For the same reason machine learning authors report precision *and* recall, and geometric reconstruction benchmarks [Seitz et al. 2006, Strecha et al. 2008, Aanæs et al. 2016] report geometric accuracy *and* completeness. For our purpose we use the percentage of rendered pixels as completeness.

7. Experimental Meta-Evaluation

In the following, we perform an evaluation of our proposed methodology with a range of experiments. In Sections 7.1 and 7.2, we demonstrate how degradations in the reconstructed model or the input data influence the computed accuracy. We show in particular that our metric fulfills the ordering criterion defined in Chapter 6. In Section 7.3, we analyze to what extent deviating from the classical, strict separation of training and test set decreases the validity of our evaluation. In Section 7.4, we discuss the relation between our methodology and the standard Middlebury MVS benchmark [Seitz et al. 2006]. In Section 7.5, we demonstrate our approach's versatility by applying it to different reconstruction representations. One possible use case for this is a 3D reconstruction benchmark open to all image-based modeling and rendering techniques, such as the one we introduce in Section 8.2. Finally, in Section 7.6, we analyze the correlation between virtual rephotography error and human judgment about visual 3D reconstruction error.

7.1. Evaluation with Synthetic Degradation

In this section we show that virtual rephotography fulfills the aforementioned ordering criterion on very controlled data. If we have a dataset's ground-truth geometry and can provide a good quality texture, this should receive zero or a very small error. Introducing artificial defects into this model decreases the model's quality, which should in turn be detected by the virtual rephotography approach.

We take Strecha et al.'s [2008] Fountain-P11 dataset, for which camera parameters and ground-truth geometry are given. We compensate for exposure differences in the images (using the images' exposure times and an approximate response curve for the used camera) and project them onto the mesh to obtain a near-perfectly colored model with vertex colors (the ground-truth mesh's resolution is so high that this is effectively equivalent to applying a high-resolution texture to the model). Rephotographing this colored model with precise camera calibration (including principal point offset and pixel aspect ratio) yields pixel-accurate rephotos.

Starting from the colored ground truth, we synthetically apply three kinds of defects to the model (Figure 7.1) to evaluate their effects on our metrics:

- Texture noise: In order to model random photometric distortions, we change vertex colors using simplex noise [Perlin 2002]. The noise parameter n_{tex} is the maximum offset per RGB channel.
- Geometry noise: Geometric errors in reconstructions are hard to model. We therefore use a simple model and displace vertices along their normal using

(a) Ground truth **(b)** Texture noise, $n_{\text{tex}} \approx 30\%$ **(c)** Geometry noise, $n_{\text{geo}} \approx 0.5\%$ **(d)** Simplif. mesh, $n_{\text{sim}} \approx 99\%$

Figure 7.1.: Fountain-P11 [Strecha et al. 2008] with various artificially introduced defects.

simplex noise. n_{geo} is the maximum offset as a fraction of the scene's extent.

- Simplification: To simulate different reconstruction resolutions, we simplify the mesh with edge collapse operations. n_{sim} is the fraction of removed vertices.

In Figure 7.2, we apply these defects with increasing strength and evaluate the resulting meshes using our method. In accordance with the ordering criterion, all difference metrics reflect the increase in noise with a corresponding increase in error. This is even the case for the $\Delta C_b + \Delta C_r$ metric, because there are no luminance differences due to different exposure since all images were exposure-adjusted before the experiment. One reason why the error does not vanish for $n_{\text{tex}} = n_{\text{geo}} = n_{\text{sim}} = 0$, is that we cannot produce realistic, local shading effects easily.

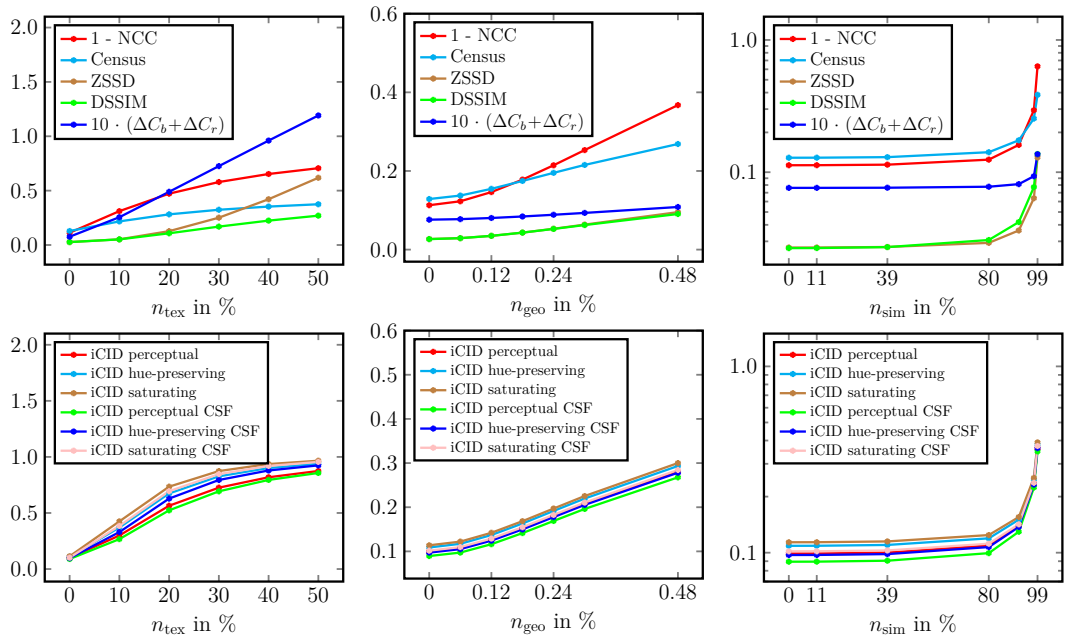


Figure 7.2.: Various error metrics for (*left to right*) texture noise, geometry noise and mesh simplification applied to Fountain-P11.

7.2. Evaluation with Multi-View Stereo Data

In the following experiment we demonstrate the ordering criterion on MVS reconstructions with more uncontrolled data. Starting from the full set of training images at full resolution, we decrease reconstruction quality by a) reducing the number of images used for reconstruction and b) reducing the resolution of the images, and show that virtual rephotography detects these degradations.

We evaluate our system with two MVS reconstruction pipelines. In the first pipeline, we generate a dense point cloud with CMVS [Furukawa et al. 2010, Furukawa and Ponce 2010], mesh the points using Poisson surface reconstruction (PSR) [Kazhdan et al. 2006], and remove superfluous triangles generated from low octree levels. We refer to this pipeline as CMVS+PSR. In the second pipeline, we generate depth maps for all views with an algorithm for community photo collections (CPCs) [Goesele et al. 2007, Fuhrmann et al. 2015] and merge them into a global mesh using a multi-scale (MS) depth map fusion approach [Fuhrmann and Goesele 2011] to obtain a high-resolution output mesh. We refer to this pipeline as CPC+MS.

We use our Citywall dataset from earlier again, which has fine details, non-rigid parts (*e.g.*, people and plants) and moderate illumination changes (it was captured over the course of two days). We apply structure from motion once to the complete dataset and use the recovered camera parameters for all subsets of training and test images. We then split our set of images into 533 training images (T) and 28 evaluation images (E). The 28 images are chosen at random using 20-fold cross-validation to rule out unfortunate splitting.

As mentioned above, we reduce the number of training images and the training image resolution to detect whether virtual rephotography detects the resulting reconstruction quality degradation. To incrementally reduce the number of training images we divide T in half three times, *i.e.*, $|T| \in \{533, 266, 133, 67\}$. To vary image resolution we use images of size $h \in \{375, 750, 1500\}$ (h being the images' shorter side length) for CMVS+PSR and $h \in \{93, 187, 375\}$ for CPC+MS.[62] Regardless of the training image height, for the evaluation we always render the reconstructed models with $h = 750$ and use a patch size of 9×9 pixels for all patch-based metrics.

Figures 7.3 and 7.4 show the following results for both the CMVS+PSR as well as the CPC+MS pipeline:

- Reconstruction **completeness** increases with increasing training set size $|T|$. The resolution h on the other hand only has a small impact on completeness.
- **1-NCC**, **Census**, **ZSSD**, **DSSIM**, and **iCID** decrease for increasing image resolution h: If we look at the box plots for a fixed $|T|$ and varying h, they are separated and ordered. Analogously, these metrics can distinguish between reconstructions with varying $|T|$ and fixed h: Reconstructions with identical

[62] The reason for using a higher resolution for CMVS+PSR is that its point clouds are much sparser than CPC+MS's depth maps. For the sake of this experiment, the different resolutions for both pipelines are irrelevant, because it is not about comparing both pipelines but about independently showing for each pipeline that virtual rephotography assigns higher errors to lower training image resolutions.

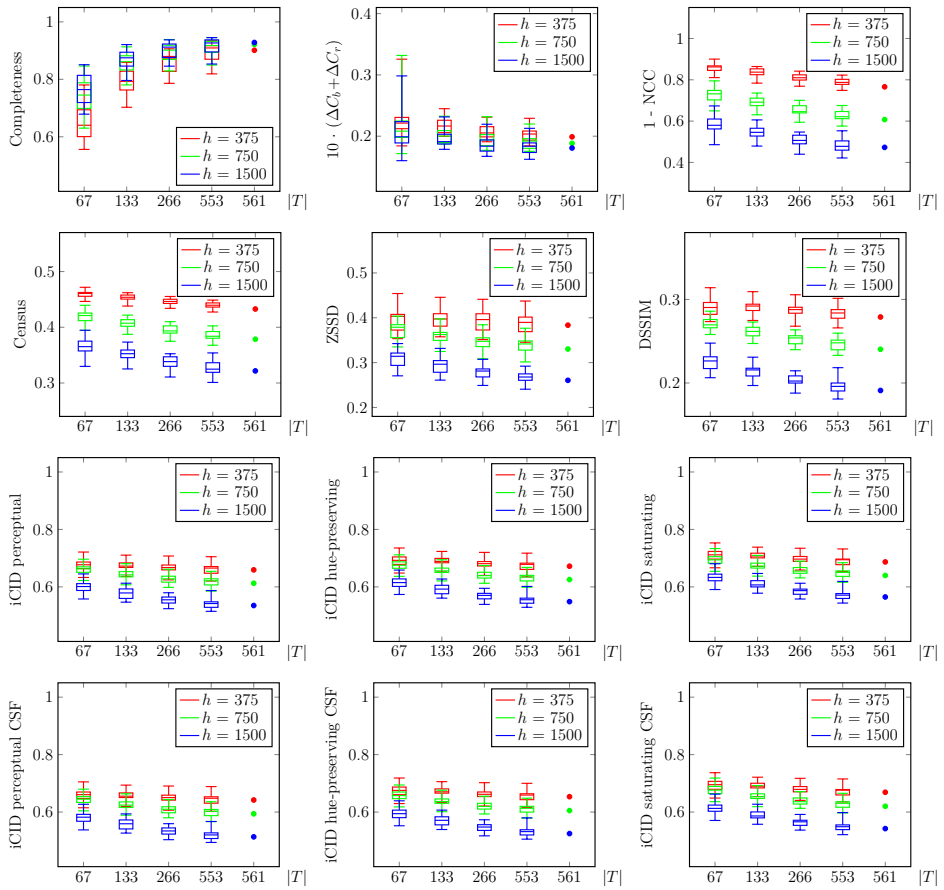


Figure 7.3.: Citywall virtual rephotography completeness and error (eleven different error metrics) for the **CMVS+PSR** reconstruction pipeline. Box plots show minimum, first quartile, median, third quartile, and maximum of 20 cross-validation iterations. $|T|$ is the training image set size and h is the images' shorter side length. Points for $|T|=561$ have been generated without cross-validation (see Section 7.3).

h and larger $|T|$ have a lower median error. These results clearly fulfill the desired ordering criterion. However, for ZSSD, DSSIM, and the iCID variants, it is less pronounced than for 1-NCC and Census: Box plots of identical $|T|$ and different h are not always clearly separated and in some cases reconstructions with identical h have a lower error for $|T| = 67$ than for $|T| = 133$.

- The pixel-based $\boldsymbol{\Delta_{C_b}+\Delta_{C_r}}$ metric fails to order reconstructions with different $|T|$ or h and invariably assigns errors of ~ 0.02. This is in contrast to the previous experiment with synthetic degradation, where the setting was significantly simpler because all images had identical illumination, camera response curves and exposure. The very same effect would occur for raw RGB difference, which Fitzgibbon et al. [2005, Figure 7d] used to evaluate their IBR method.

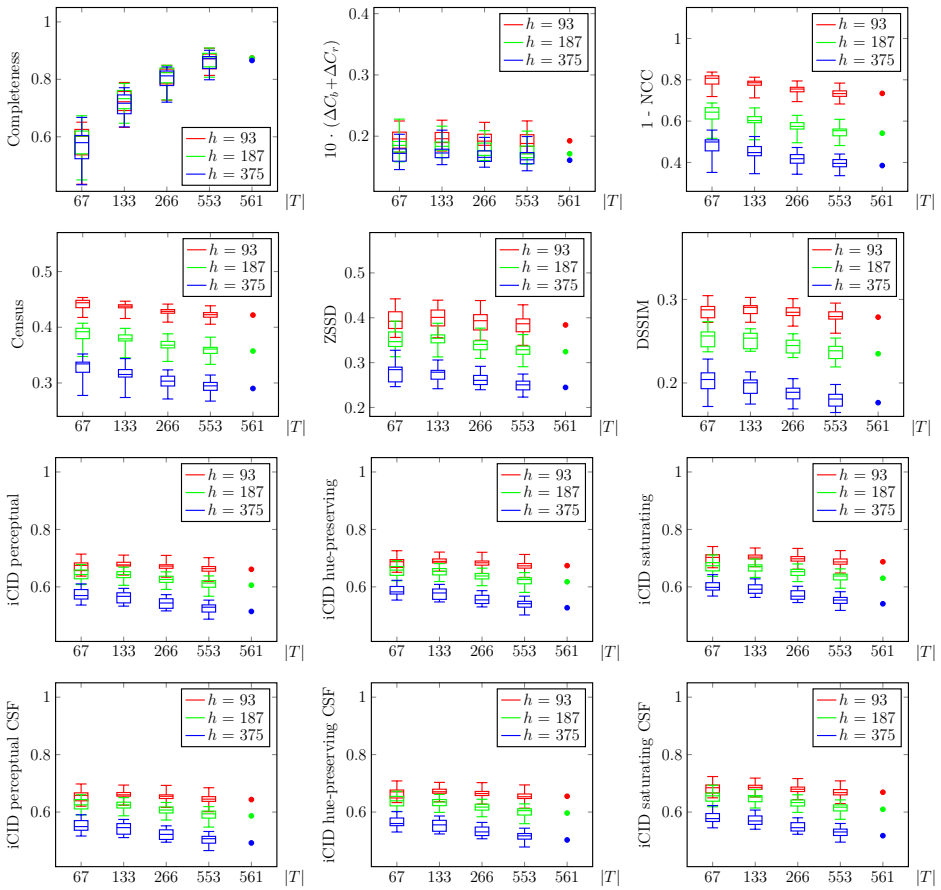


Figure 7.4.: Citywall virtual rephotography completeness and error (eleven different error metrics) for the **CPC+MS** reconstruction pipeline. Box plots show minimum, first quartile, median, third quartile, and maximum of 20 cross-validation iterations. $|T|$ is the training image set size and h is the images' shorter side length. Points for $|T| = 561$ have been generated without cross-validation (see Section 7.3).

7.2.1. Limitations

Here we shortly discuss two additional datasets that we evaluated analogously to the Citywall. For these datasets, our metrics' properties (as discussed in the previous section) start to break, because the reconstructions of these datasets are too flawed in many cases.

Regarding the Kopernikus Statue dataset (334 images) (see Figure 7.5, left), many of the reconstructions contain superfluous geometry from the surface reconstruction due to a lot of foreground clutter (mostly plants). The 1-NCC error (Figure 7.6, row 2, columns 1 and 2) correctly ranks reconstructions with a larger training set size $|T|$ better. Additionally, it consistently ranks reconstructions with medium image resolution better than those with low resolution but struggles with distinguishing between medium and high resolution.

Figure 7.5.: *Left:* Kopernikus Statue dataset. *Right:* Hass Statue dataset.

The reconstructions of the Hass Statue dataset (see Figure 7.5, right) are often seriously flawed because there are only a few (49) input images available. We could not remove many images without completely destroying the reconstruction. Thus, we varied the training set size between $|T| = 23$ and 47 only. In this setting, the 1-NCC error (Figure 7.6, row 2, columns 3 and 4) is unable to distinguish between reconstructions with different $|T|$. However, it orders reconstructions with different image resolutions h correctly, although the separation is less pronounced compared to the Citywall. As with the Citywall previously, the $\Delta C_b + \Delta C_r$ error assigns a more or less constant error to all reconstructions. Both datasets demonstrate that virtual rephotography struggles with datasets that do not reconstruct well.

7.3. Disjointness of Training and Test Set

In Section 2.3 we discussed Kang et al.'s [2006] IBR continuum. Methods at the bottom end of the continuum that produce a static geometry model with a static texture globally explain all input images as a whole. Contrarily, methods further to the top of the continuum such as light fields, view-dependent geometry, or view-dependent texture can be seen as a set of many local descriptions of the world. In view-dependent methods, the rendering algorithm chooses at rendering time which subset of these local descriptions it uses for synthesizing the virtual view, depending on the virtual view parameters. It is therefore imperative for view-dependent methods to separate training and test images, since they could otherwise simply display the known test images for evaluation and receive a perfect score. This is the reason why we randomly split the set of images into disjoint *training* and *test sets* and use cross-validation. However, using all available images for reconstruction typically yields the best reconstruction results and it may therefore be undesirable to "waste" perfectly good images by solely using them for the evaluation. This is particularly relevant for datasets which only contain a few images to begin with and for

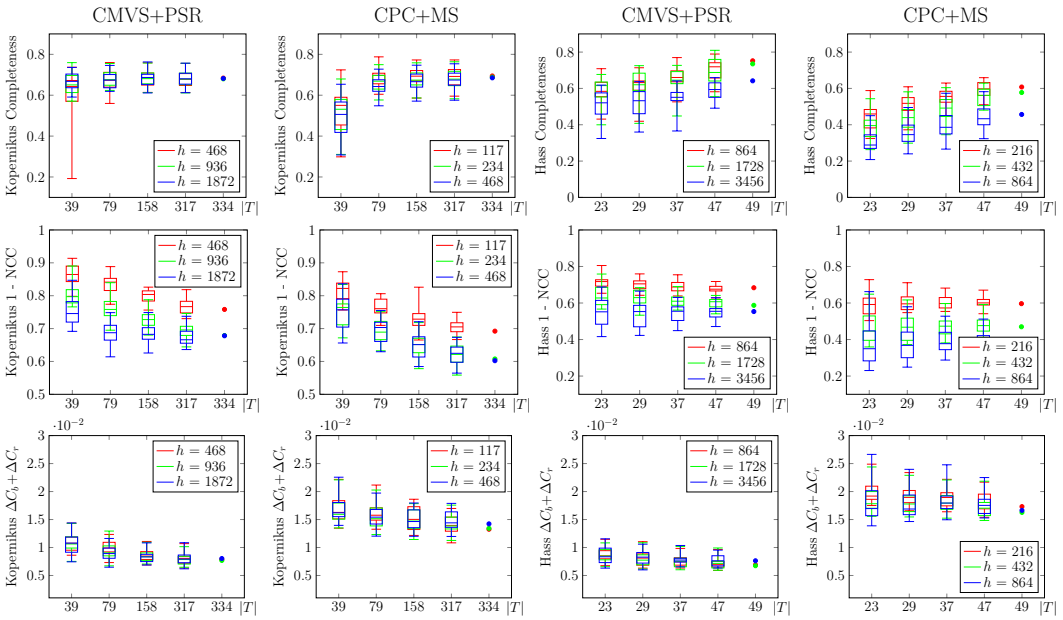


Figure 7.6.: Rephoto completeness, 1-NCC, and ΔC_b+ΔC_r error for the CMVS+PSR (*columns 1 and 3*) and the CPC+MS (*columns 2 and 4*) pipeline on the Kopernikus Statue (*columns 1 and 2*) and Hass Statue (*columns 3 and 4*). Box plots show minimum, first quartile, median, third quartile, and maximum of 20 cross-validation iterations.

which reconstruction may fail when removing images. Moreover, even though cross-validation is an established statistical tool, it is very resource- and time-consuming.

We now show for the CPC+MS and CMVS+PSR pipelines, which are both at the IBR continuum's bottom end, that evaluation can be done without cross-validation with no significant result changes. On the Citywall we omit cross-validation and obtain the data points for $|T| = 561$ in Figures 7.3 and 7.4. Most $|T| = 561$ data points have slightly smaller errors than the median of the largest cross-validation experiments ($|T| = 533$), which is reasonable since the algorithm has slightly more images from which to reconstruct. Neither CMVS+PSR nor CPC+MS seem to overfit the input images. We want to point out that, although this may not be generally applicable, it seems safe not to use cross-validation for the scene representations with static geometry and static texture used here. In contrast, approaches like the unstructured Lumigraph [Buehler et al. 2001] will just display the input images and thus break an evaluation where the test set is not disjoint from the training set.

7.4. Comparison with Geometry-based Benchmark

The Middlebury MVS benchmark [Seitz et al. 2006] provides images of two objects, Temple and Dino, together with accurate camera parameters. We now investigate the correlation between virtual rephotography and Middlebury's geometric evalua-

Table 7.1.: Correlation of geometric and rephoto error for all patch-based metrics. Bold font indicates the per-row maximum, *i.e.*, strongest correlation.

Dataset	99.9% signific. level	99.99% signific. level	Correlation ρ of geometric error and 1-NCC	Census	ZSSD	DSSIM	iCID per.	iCID hue.	iCID sat.	iCID per. CSF	iCID hue. CSF	iCID sat. CSF	$\Delta C_b+\Delta C_r$
TempleRing	$\lvert\rho\rvert>0.49$	$\lvert\rho\rvert>0.57$	0.63	**0.67**	0.52	0.60	0.57	0.57	0.57	0.57	0.57	0.57	0.30
DinoRing	$\lvert\rho\rvert>0.50$	$\lvert\rho\rvert>0.58$	0.69	**0.85**	0.32	0.44	0.41	0.40	0.39	0.40	0.39	0.38	-0.12

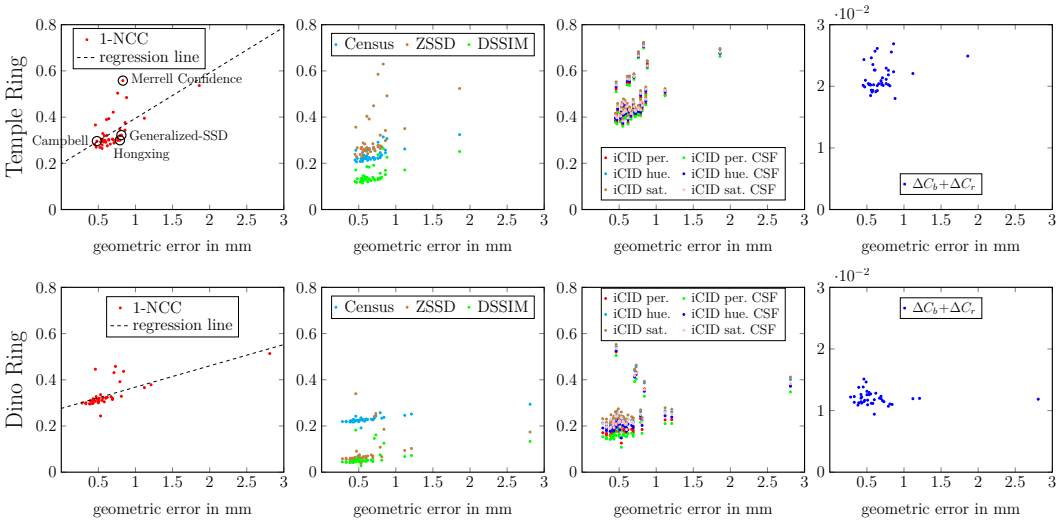


Figure 7.7.: Rephoto error against geometric error for 41 TempleRing (*top row*) and 39 DinoRing (*bottom row*) Middlebury benchmark submissions.

tion using the TempleRing and DinoRing variants of the datasets, which are fairly stable to reconstruct and are most frequently submitted for evaluation. With the permission of their respective submitters, we analyzed 41 TempleRing and 39 DinoRing submissions with publicly available geometric error scores. We transformed the models with the ICP alignment matrices obtained from the Middlebury evaluation, removed superfluous geometry below the model base, textured the models [Waechter et al. 2014b] (see Section 3), rendered them, and evaluated them.

Analyzing the Pearson correlation between rephoto error and the geometric error reported by the Middlebury benchmark yields the correlation coefficients shown in Table 7.1. Given the number of data points for the correlation analysis on the TempleRing and DinoRing – 41 and 39, respectively – correlation coefficients with absolute value greater than 0.49 and 0.50, respectively, are significant at a level of 99.9 %, *i.e.*, there is a chance of less than 0.1 % for rephoto and geometric error to be uncorrelated if the correlation coefficient is above 0.49 or 0.50, respectively. 1-NCC and Census both seem to be strongly correlated to the geometric error, the other patch-based metrics exhibit a weaker correlation, and $\Delta C_b + \Delta C_r$ may be uncorrelated.

Figure 7.7 shows scatter plots of rephoto error against geometric error. All patch-

based metrics (1-NCC, Census, ZSSD, DSSIM, iCID) have a similar data series layout and seem to be correlated with geometric error.

Despite the correlation between, *e.g.*, 1-NCC and geometric error, we will have a closer look at some outliers in the top left graph of Figure 7.7: *E.g.*, the geometric errors for the methods "Merrell Confidence" [Merrell et al. 2007a] and "Generalized-SSD" [Calakli et al. 2012] are similar, whereas their 1-NCC errors differ strongly. Conversely, the geometric errors of "Campbell" [Campbell et al. 2008] and "Hongxing" [Hongxing et al. 2010] are very different, but their 1-NCC errors are almost identical. We showed renderings of these models in Figure 5.2 and discussed the visual dissimilarity of 5.2a and 5.2b and the similarity of 5.2c and 5.2d on page 83. Apparently, visual accuracy explains why the rephoto error does not follow the geometric error for the pairs Merrell Confidence vs. Generalized-SSD and Campbell vs. Hongxing. Clearly virtual rephotography captures aspects that complement the purely geometric Middlebury evaluation.

7.5. Different Reconstruction Representations

One major advantage of our approach is that it handles arbitrary image-based modeling and rendering (IBMR) representations since all can be rendered from novel views. We demonstrate this using the Castle Ruin dataset (286 images) and five different representations, four of which are shown in Figure 5.1:

- Point cloud: Multi-view stereo algorithms output oriented point clouds that can directly be rendered with surface splatting [Zwicker et al. 2001]. As splat radius we use the local point cloud density (three times the distance to a point's third nearest neighbor).
- Static geometry with vertex colors: This is the result of a surface reconstruction technique run on a point cloud.
- Static geometry with static texture: Meshes can be textured using the input images. We use the texturing algorithm from Chap. 3 [Waechter et al. 2014b].
- View-dependent geometry with view-dependent texture: Using depth maps as view-dependent geometry proxies, we reproject all images into the novel view and render color images as well as per-pixel weights derived from a combination of angular error [Buehler et al. 2001] and TS3 error [Kopf et al. 2014]. We then fuse the color and weight image stack by computing the weighted per-channel median of the color images.
- Static geometry with view-dependent texture: We use the previous algorithm and replace the local geometric proxies with a globally reconstructed mesh.

Except for the fourth, we base all representations on the 3D mesh reconstructed with the CPC+MS pipeline. For the fourth representation, we reconstruct per-view geometry (*i.e.*, depth maps) for each input image using SGM [Hirschmüller 2008]. For the fourth and fifth representation, we perform leave-one-out cross-validation; the other representations are evaluated without cross-validation. Further, for better

Table 7.2.: Rephoto errors for different representations of the Castle Ruin (*cf.* Figure 5.1). Bold font indicates the per-column minimum, *i.e.*, best-rated representation.

Representation	Comp.	1-NCC	Census	ZSSD	DSSIM	iCID per.	iCID hue.	iCID sat.	iCID per. CSF	iCID hue. CSF	iCID sat. CSF	ΔC_b+ΔC_r
Static geom., static tex.	0.66	**0.29**	**0.21**	**0.22**	**0.16**	**0.48**	**0.50**	**0.51**	**0.46**	**0.48**	**0.50**	0.017
Static geom., view-dep. tex.	0.66	0.39	0.26	0.24	0.19	0.52	0.54	0.55	0.50	0.52	0.53	0.017
Static geom., vertex colors	0.66	0.47	0.32	0.26	0.22	0.58	0.59	0.61	0.56	0.57	0.59	**0.016**
Splatted point cloud	0.67	0.51	0.33	0.27	0.23	0.59	0.60	0.62	0.57	0.59	0.60	**0.016**
View-dep. geom. & tex.	0.66	0.62	0.36	0.46	0.28	0.69	0.70	0.71	0.67	0.68	0.70	0.022

comparability we restricted the evaluation of the fourth and fifth method to those parts of the rephotos where the CPC+MS depth maps indicate valid depth.

Table 7.2 shows the resulting errors. All patch-based metrics rank the scene representations in the following order (from best to worst): static geometry with static texture, static geometry with view-dependent texture, static geometry with vertex colors, splatted point cloud, view-dependent geometry and texture. This order is consistent with our visual intuition when manually examining the representations' rephotos (compare also the different rephotos in Figure 5.1): The static mesh with static texture is ranked best. The static mesh with view-dependent texture, which was ranked second, could in theory produce lower errors but our implementation does not perform luminance adjustments at patch seams (in contrast to our texturing algorithm), which virtual rephotography detects as erroneous. The point cloud has almost the same error as the static mesh with vertex colors since both are based on the same mesh and thus contain the same geometry and color information. Only their rendering algorithms differ, as the point cloud rendering algorithm discards vertex connectivity information. The view-dependent geometry and view-dependent texture algorithm being ranked worst may seem unintuitive, but our implementation suffers from strong artifacts caused by imperfectly reconstructed planar depth maps used as geometric proxies. These cause strong error responses.

In contrast to the patch-based metrics, the pixel-based ΔC_b+ΔC_r error does not correspond with our intuition and even ranks the point cloud as the best.

We note that our findings only hold for our implementations and parameter choices and no representation is fundamentally superior to others.

7.6. User Study: Correlation with Human Judgment

In order to not only put virtual rephotography in relation to our own, potentially subjective visual intuition, we conducted a user study to shed light on the correlation between virtual rephotography scores and human judgment.

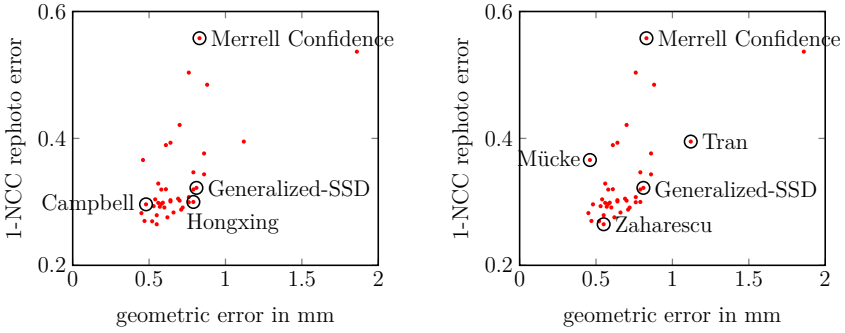
Setup: We performed a user study to determine the correlation between virtual rephotography and human ratings of visual reconstruction accuracy. We used two 1080p monitors which we calibrated to the CIE D65 white point and $160\,\mathrm{cd\,m^{-2}}$ to ensure similar luminance and color display. Out of our 25 study participants, 9 had

Figure 7.8.: The interface of our user study. *Top:* A reference photo. *Bottom:* A dual screen setting with five rephotos from five different scene representations that users should drag and drop into the order of similarity to the reference. (Best viewed on screen.)

a background in 3D reconstruction or computer graphics (called "3D experts" in the following) and 16 did not ("non-experts" in the following). Prior to the main study, each participant underwent a tutorial explaining the task and letting them test the interface on a small example dataset.

Users were first shown an input photo (top image in Figure 7.8), and clicking revealed multiple rephotos of the same scene from the same viewpoint but reconstructed and rendered by different means (bottom image in Figure 7.8). Those multiple rephotos were shown simultaneously on the two monitors and at the same resolution at which the virtual rephotography framework evaluated them. Users then had to drag and drop the rephotos into the order of which rephoto they perceived as most or least similar to the input photo. They were instructed not to take black regions in the rephotos into account. No time limit was imposed on them and they could toggle back and forth between photo and rephotos at any time to compare them precisely. After users were satisfied with the order of the rephotos, they could continue with the next view. The different viewpoints of each scene were shown in random temporal order and the different rephotos per viewpoint were shown in random spatial on-screen order. Based on feedback from a prior sandbox test of our study, we determined that five rephotos is more or less the maximum that a participant can judge simultaneously.

Datasets: The first dataset we showed participants was the *Castle Ruin* and its five different reconstruction representations from Section 7.5. We chose it to back



(a) Middlebury-4 **(b)** Middlebury-5

Figure 7.9.: The positions of the user study datasets Middlebury-4 (*left*) and Middlebury-5 (*right*) in the scatter plot of 1-NCC rephoto error vs. geometric error.

(a) "Zaharescu" [Zaharescu et al. 2007] **(b)** "Generalized-SSD" [Calakli et al. 2012] **(c)** "Mücke" [Mücke et al. 2011] **(d)** "Tran" [Tran and Davis 2006] **(e)** "Merrell Confidence" [Merrell et al. 2007a]

Figure 7.10.: The five submissions in the Middlebury-5 dataset.

up our claims about rephoto errors being "consistent with our visual intuition when manually examining the representations' rephotos" that we made on page 104. Further, it is a rather complex outdoor dataset with changing illumination. We reduced the dataset from its full 285 views to 142 randomly selected views.

The second dataset we let our participants judge consists of the four Middlebury TempleRing submissions which we showed and discussed earlier (Figures 5.2 and 7.7): "Merrell Confidence", "Generalized-SSD", "Campbell", and "Hongxing". Their positions in the scatter plot of 1-NCC rephoto error vs. geometric error are shown again in Figure 7.9, left. We call this dataset *Middlebury-4* in the following.

Since Campbell, Hongxing and Generalized-SSD have similar rephoto error and Hongxing, Generalized-SSD and Merrell Confidence have similar geometric error, we also created a third dataset with a different selection of TempleRing submissions: "Mücke" [Mücke et al. 2011], "Zaharescu" [Zaharescu et al. 2007], "Generalized-SSD" [Calakli et al. 2012], "Merrell Confidence" [Merrell et al. 2007a], and "Tran" [Tran and Davis 2006], which are shown in Figure 7.9, right and Figure 7.10. We selected

these submissions because they are spread out very far in the scatter plot and because geometric error and rephoto error rank them in a very different order (from best to worst): Mücke, Zaharescu, Generalized-SSD, Merrell Confidence, Tran for the geometric error vs. Zaharescu, Generalized-SSD, Mücke, Tran, Merrell Confidence for 1-NCC rephoto error. We call this dataset *Middlebury-5* in the following.

Results: For all decisions, the average user took about 71 min for the Castle Ruin, 18 min for Middlebury-4 and 17 min for Middlebury-5.[63] Since the initial order of rephotos is random, when averaging over all views that a user judged, the relative frequency of a rephoto from initial position i ending up at position j after the user is done ordering the rephotos should ideally be $\frac{1}{\text{number of rephotos}}$ and $\sum_i p(\text{image } i \text{ ending up at position } i)$ should be 1. Based on this analysis, we excluded the results of three users who left rephotos at their initial position significantly more often than this from our analysis.[64] Further, we excluded the results of two users who took a lot less time for their decisions than all others. These two exclusion criteria are agnostic of the actual decisions that the users made in the study. We thereby obtained a total of $(25-2-3)\cdot(142+47+47)$ rankings.

The machine-computed virtual rephoto *scores* and the human *rankings* are not directly comparable. We therefore converted the machine's per-view per-reconstruction scores into per-view per-reconstruction ranks. For the human (per-user, per-view, per-reconstruction) ranks, we eliminated the user as a variable of the data by computing the mode (*i.e.*, most common value) over all users, also giving us per-view per-reconstruction ranks. We then computed Spearman's rank correlation between machine and human ranks and obtained the correlations shown in Table 7.3. Interestingly, we found almost no difference between 3D experts and non-experts in the rank correlations and the decision times. On the Castle Ruin, 1-NCC and Census exhibit the strongest and $\Delta C_b{+}\Delta C_r$ the weakest correlation with human judgment.

Table 7.3.: Rank correlations between human judgment and rephoto error for various image difference metrics. Bold font indicates the per-row maximum, *i.e.*, strongest correlation.

Dataset	User group	1-NCC	Census	ZSSD	DSSIM	iCID per.	iCID hue.	iCID sat.	iCID per. CSF	iCID hue. CSF	iCID sat. CSF	$\Delta C_b{+}\Delta C_r$
Castle Ruin (see Fig. 5.1 and Tab. 7.2)	All	0.630	**0.633**	0.379	0.465	0.385	0.381	0.369	0.390	0.381	0.366	0.169
	3D experts	0.634	**0.639**	0.357	0.459	0.379	0.376	0.365	0.385	0.376	0.363	0.158
	non-experts	0.626	**0.627**	0.396	0.469	0.389	0.384	0.373	0.394	0.385	0.369	0.178
Middlebury-4 (Fig. 7.9, left)	All	0.519	0.511	0.568	0.547	0.481	0.470	0.476	0.490	0.495	0.498	**0.629**
	3D experts	0.527	0.514	0.556	0.558	0.482	0.470	0.478	0.487	0.487	0.500	**0.623**
	non-experts	0.513	0.509	0.578	0.539	0.479	0.471	0.475	0.492	0.501	0.496	**0.634**
Middlebury-5 (Fig. 7.9, right)	All	0.870	0.820	**0.891**	0.876	0.884	0.884	0.884	**0.891**	0.889	0.889	0.760
	3D experts	0.867	0.817	**0.889**	0.873	0.882	0.882	0.882	**0.889**	0.886	0.886	0.762
	non-experts	0.873	0.823	**0.892**	0.877	0.886	0.886	0.886	**0.892**	0.890	0.890	0.759

[63] We attribute the lower time for Middlebury-5 (compared to Middlebury-4) to the better discriminability of the datasets in Middlebury-5.

[64] $\sum_i p(\ldots) = 1.67$, 1.49, and 1.48, respectively. Some participants had a $\sum_i p(\ldots)$ as low as 0.99 or 1.08.

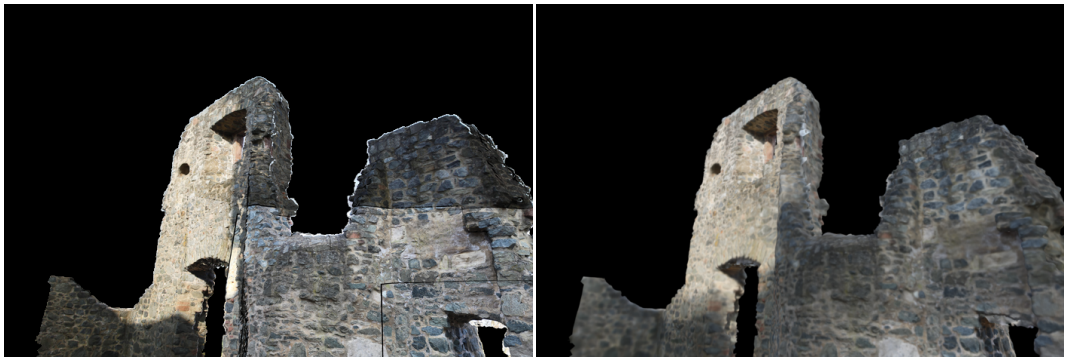


Figure 7.11.: Rephotos of a 3D reconstruction of Frankenstein Castle rendered with image-based rendering (*left*) and vertex colors (*right*).

There are a variety of potential reasons why the correlation between the users and, *e.g.*, 1-NCC is smaller than 1, including the following: In a questionnaire that users filled out directly after the study, we showed them the two renderings shown in Figure 7.11, of which the left one is very detailed but has image regions with different luminance and sharp transitions between these regions, and the right is globally consistent but blurry. We asked them which of the two they deemed more similar to the reference image and 37 % of all users preferred the right image. While this is by far not a clear preference for one or the other, it indicates that humans at least partially take global effects such as consistency into account when judging image similarity.[65] In contrast, all image difference metrics we employed are strictly local in nature (since they are restricted to their patch size) and almost always prefer the sharp image, because sharp regions have a small local error, only the relatively small transition regions have a high local error, and the large majority of rectangular patches of the difference metrics does not overlap with any transition region. It would clearly be interesting to analyze global image difference metrics with respect to their correlation to human judgment, but to our knowledge, those do not exist in the current state of research in image difference metrics.

ZSSD, DSSIM, iCID, and especially $\Delta C_b + \Delta C_r$ have a surprisingly strong correlation with human judgment on Middlebury-4 and Middlebury-5. We suppose that this is due to the Middlebury Temple being a very controlled dataset (similar to the Strecha dataset from Section 7.1 where $\Delta C_b + \Delta C_r$ also performed well).

Finally, we take Middlebury-5 and compare the correlation of humans vs. rephoto error with the correlation of humans vs. geometric error: We take the user study results, eliminate the user and the view as a variable of the data by com-

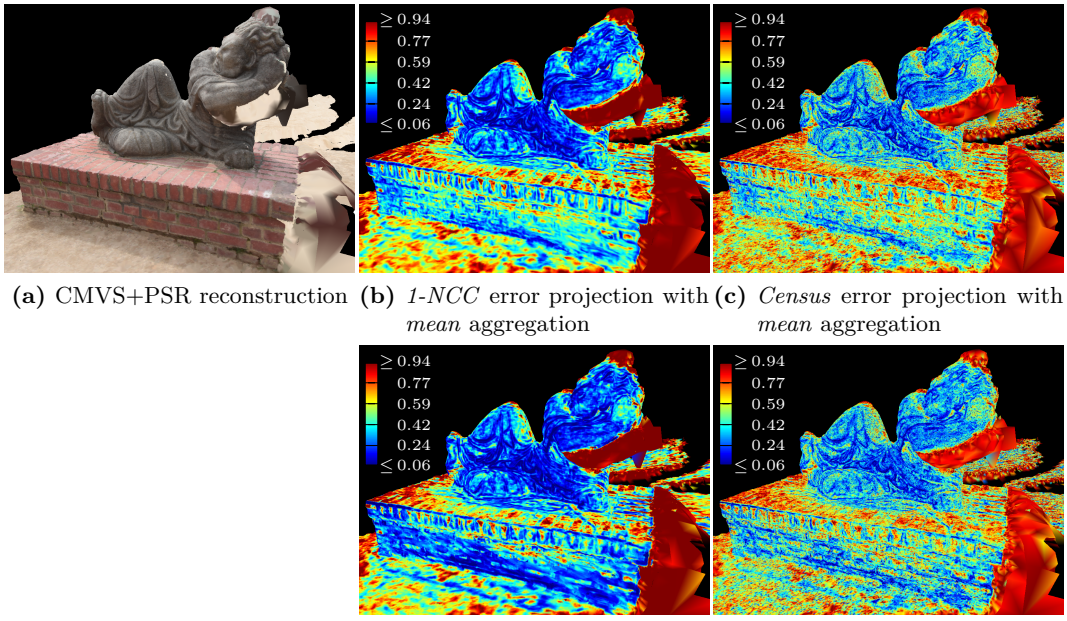
[65] This may be related to the fact that in IBR blending artifacts (one virtual pixel being blended from multiple source pixels that do not correspond due to incorrect depth values) are more noticeable than popping artifacts (no blending; when the observer moves through the scene and a destination image region suddenly switches from one source image to another, objects "jump") [Mustafa et al. 2012]. Apparently, discontinuities can be disturbing both temporally and spatially, so much so that clearly blurry but discontinuity-free results may be preferable.

puting the mode over both, and obtain the following order (from best to worst): Zaharescu, Generalized-SSD, Mücke, Tran, Merrell Confidence. We also turn the geometric Middlebury errors into ranks, which gives us the order Mücke, Zaharescu, Generalized-SSD, Merrell Confidence, Tran. Taking the 1-NCC rephoto scores and converting them to per-reconstruction ranks gives us the order Zaharescu, Generalized-SSD, Mücke, Tran, Merrell Confidence. We thereby obtain a rank correlation of 0.6 for humans vs. geometric error and 1 for humans vs. rephoto error. Admittedly, this analysis is not as fine-grained as the previous ones because we only have one Middlebury score per reconstruction (as opposed to per view, per reconstruction) and therefore had to break human ranks and rephoto ranks down to per-reconstruction ranks, which gives us only 5 data points for the correlation computation. Further, this is comparing apples with oranges to some degree, since humans and rephoto error judge 2D projections, whereas geometric error only judges geometry. Nevertheless, we keep up our initial claim that we made on pages 81f: There are reconstruction properties that are somewhat orthogonal to geometry and geometric accuracy is not necessarily a good predictor of visual accuracy.

8. Analyzing Reconstructions with Virtual Rephotography

In this chapter, we present two important applications of virtual rephotography: In Section 8.1, we use virtual rephotography to locally highlight defects in MVS reconstructions, which can, *e.g.*, be used to guide users to regions where additional images need to be captured to improve the reconstruction. In Section 8.2, we introduce a general benchmark for all image-based reconstruction and rendering techniques.

8.1. Error Localization on Geometric Reconstructions

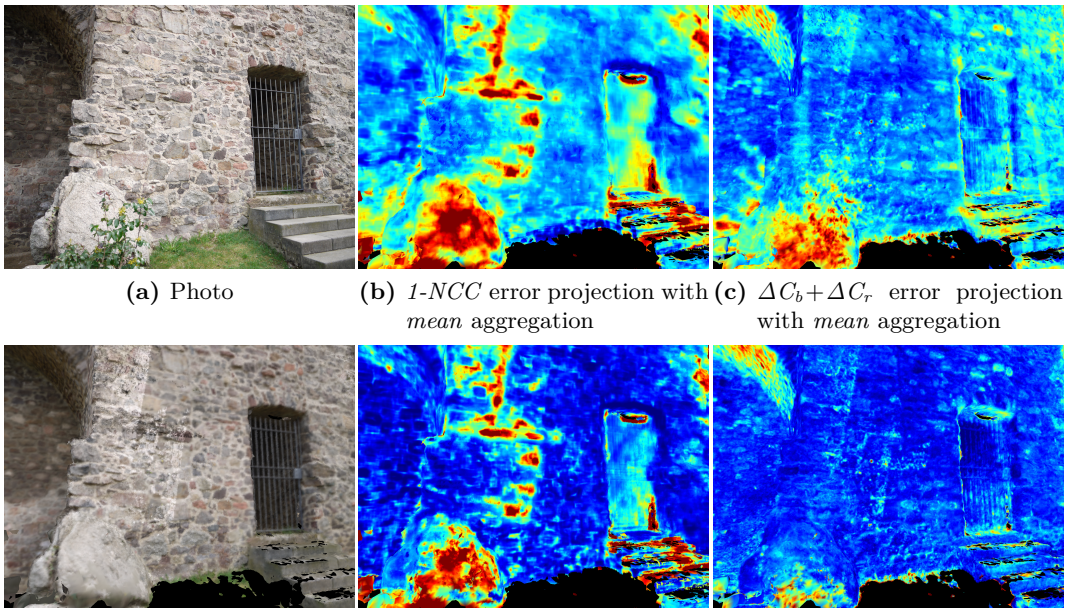


(a) CMVS+PSR reconstruction **(b)** *1-NCC* error projection with *mean* aggregation **(c)** *Census* error projection with *mean* aggregation

(d) *1-NCC* error projection with *median* aggregation **(e)** *Census* error projection with *median* aggregation

Figure 8.1.: Hass Statue reconstruction and various error projections.

If the evaluated reconstruction contains an explicit geometry model (which is not the case, *e.g.*, for a traditional light field [Levoy and Hanrahan 1996]), we can project the computed error images onto the model to visualize reconstruction defects directly. We aggregate multiple error images projected to the same location by computing the mean or the median. To improve the visualization contrast, we normalize all errors between the 2.5% and 97.5% percentile to the range $[0, 1]$, clamp



(a) Photo

(b) *1-NCC* error projection with *mean* aggregation

(c) $\Delta C_b + \Delta C_r$ error projection with *mean* aggregation

(d) Rephoto with color defects on the wall to the door's left

(e) *1-NCC* error projection with *median* aggregation

(f) $\Delta C_b + \Delta C_r$ error projection with *median* aggregation

Figure 8.2.: A color defect on a Citywall reconstruction is detected by the 1-NCC error (*middle column*) but not by the $\Delta C_b + \Delta C_r$ error (*right column*).

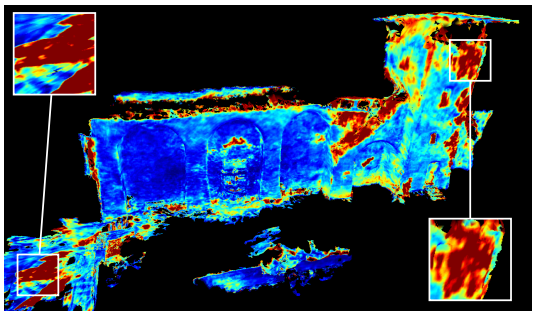
errors outside the range, and map all values to colors using the "jet" color map.

Figure 8.1 shows rephoto error projections on a reconstruction of the Hass Statue for 1-NCC and Census error and with mean and median aggregation. They highlight blob-like Poisson surface reconstruction [Kazhdan et al. 2006] artifacts behind the bent arm, to the right of the pedestal, and above the head. In a less pronounced manner, they highlight the ground and the pedestal's top which were photographed at acute angles. We note that 1-NCC localizes reconstruction errors relatively compactly whereas Census's errors are more spread out across the whole reconstruction. Similarly, median aggregation localizes errors more compactly than mean aggregation, which can likely be attributed to the median's better outlier robustness.

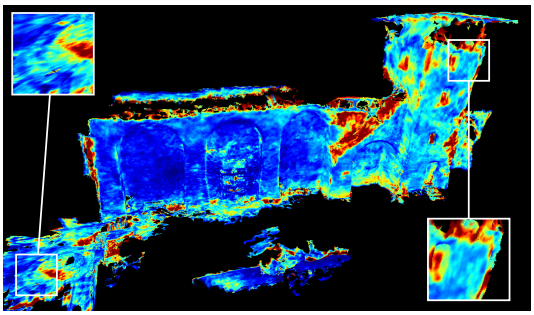
Figure 8.2 shows that color defects in a reconstruction that result from combining images with different exposure are detected by the 1-NCC error. In contrast, the $\Delta C_b + \Delta C_r$ error fails to detect these defects because it cannot distinguish between per-pixel luminance changes due to noise and medium- or large-scale changes due to illumination/exposure differences. Figure 8.3 shows a textured Citywall model and its 1-NCC error projections. The error projections properly highlight hard-to-reconstruct geometry such as grass on the ground or the tower's upper half, which was only photographed from a distance. It also strongly highlights mistextured parts, such as tree branches on the tower or the pedestrian on the ground in the left. In the right column, we show a variant of the Citywall where photometric

(a) Citywall reconstruction where a pedestrian and a tree have been used as texture.

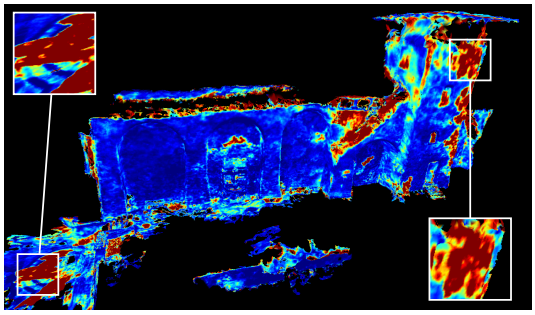
(b) A photo-consistency check (Section 3.3.2.2) removes the pedestrian and the tree.



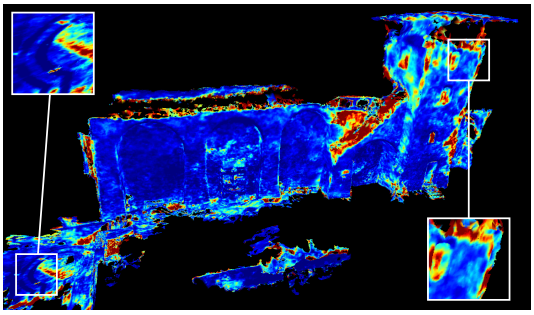
(c) 1-NCC error projection for (a) with *mean* aggregation.



(d) 1-NCC error projection for (b) with *mean* aggregation.



(e) 1-NCC error projection for (a) with *median* aggregation.



(f) 1-NCC error projection for (b) with *median* aggregation.

Figure 8.3.: Citywall reconstruction without (*left*) and with (*right column*) outlier removal.

outliers such as the pedestrian and the tree have automatically been removed by the texturing algorithm (Section 3.3.2.2). The error projection nicely tracks the outlier removal by turning from red to blue. Like in Figure 8.1, median aggregation localizes errors more compactly than mean aggregation in Figures 8.2 and 8.3.

8.2. Image-Based Modeling and Rendering Benchmark

Based on our proposed framework we built an IBMR benchmark which is available online at `https://ibmr-benchmark.gcc.informatik.tu-darmstadt.de`. As with the normal virtual rephotography evaluation procedure, it provides training images

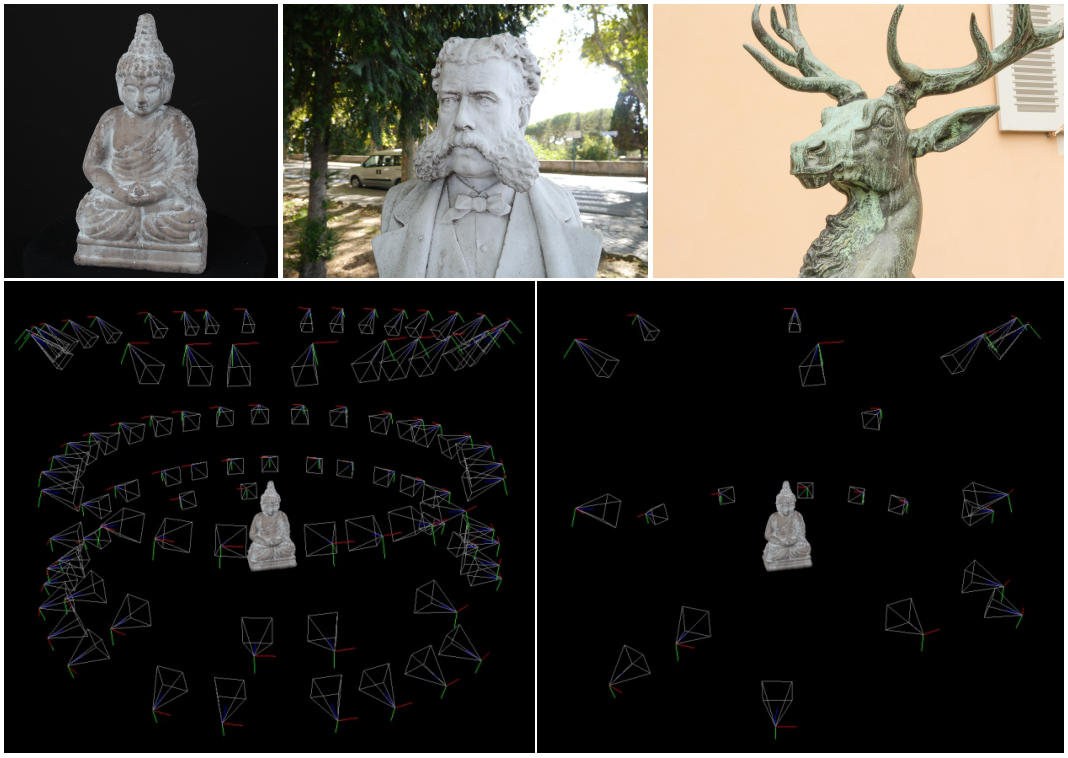


Figure 8.4.: *Top row:* Our three datasets: Buddha (91 training and 22 test images), Mattia (296 training and 73 test images), and Deer (312 training and 78 test images). *Bottom row:* The positions of the Buddha dataset's training (*left*) and test images (*right*).

with known camera parameters, asks submitters to render images of their IBMR scenes using camera parameters of secret test images, and computes completeness and 1-NCC error scores for the test images.

Our three scenes can be seen in Figure 8.4 (top row). The first dataset, *Buddha*, is a controlled indoor scene with a black background and constant, diffuse lighting. Images were captured on four horizontal rings around the figurine with the camera on a tripod and the figurine rotated on a turntable. The second dataset, *Mattia*, is an outdoor dataset of a whitish bust with weathering effects, a lot of background clutter, and strong background lighting in some images. The third dataset, *Deer*, is a bronze statue on a pedestal, with a lot of fine details due to corrosion. At least in simple surface reconstruction approaches such as Poisson surface reconstruction [Kazhdan et al. 2006], the relatively thin deer antlers cause problems in combination with scene clutter from noisy depth maps. Because of the length of the deer's torso, close-up photos of it can have strong depth-of-field blur in background parts. We therefore took them with a very small aperture and a long exposure and downscaled the images once to minimize noise. Both Mattia and the Deer have images with drastic scale differences. Benchmark submitters must therefore take this into account (*e.g.*, in their surface reconstruction or texturing algorithms) to achieve good scores.

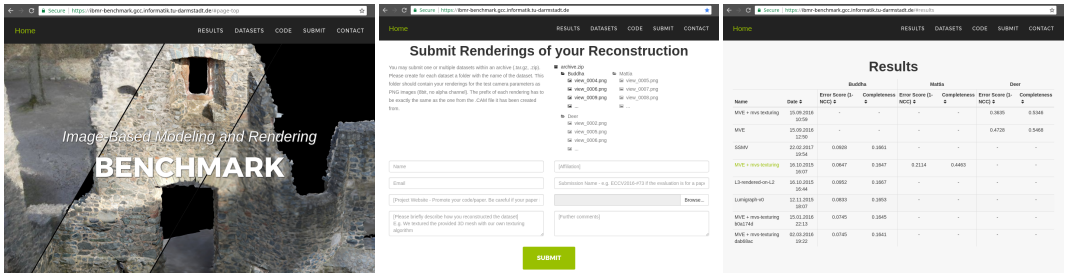


Figure 8.5.: *Left to right:* Benchmark website's welcome screen, submission form, and result table.

When choosing the test images, we ensured with a greedy sampling algorithm that the test images are relatively evenly distributed among all images. An example can be seen at the bottom right of Figure 8.4.

The publicly available data for each dataset are images and camera parameters (both in a general format as well as MVE format [Fuhrmann et al. 2015]) for the training images and camera parameters for the test images. Further, we provide a range of shortcuts in the benchmark in order to help researchers that focus only on individual reconstruction aspects instead of a complete pipeline: Starting from our own reconstruction pipeline that produces IBMR representations with static geometry and static texture, we provide all intermediate results and algorithm implementations. Specifically, we provide depth maps, a good surface mesh (that could also be used for an unstructured Lumigraph), MVE's code for multi-view stereo [Goesele et al. 2007] and surface reconstruction [Fuhrmann and Goesele 2014], and our own texturing code. Further, we also provide code to render static mesh models and code to compare photos and rephotos with 1-NCC so submitters can inspect what our benchmark does internally. Submitters can of course also choose a validation dataset from the training images and use our rendering code in order to, *e.g.*, verify whether they implemented our camera parameter conventions correctly.

Submitters must submit a complete .zip file with renderings for all test views of at least one dataset. Submissions are handled automatically via a web framework (see the UI in Figure 8.5, center): After providing some basic information such as a name and a contact e-mail address, the .zip-file can be uploaded, and the system sends a notification to the submitter and the benchmark administrators and pauses the evaluation until one of the administrators starts it.[66] After it has been started, the submission is automatically evaluated and the resulting score is posted in the results section of the benchmark page (see Figure 8.5, right).

We believe that our unified benchmark will be fruitful since it provides a common base for image-based modeling and rendering methods from both the graphics and the vision community. It opens up novel and promising research directions by "allow[ing] us to measure progress in our field and motivat[ing] us to develop better algorithms" [Szeliski 1999].

[66] This prevents submission spamming with which submitters could even infer the ground truth.

9. Conclusions

In Part I of this thesis we demonstrated how to efficiently generate high-quality textured models from large real-world 3D reconstructions. After obtaining a 3D reconstruction of some sort, evaluating its quality is of great importance. In Part II we proposed an evaluation approach focusing on the rendered quality of a reconstruction, or more precisely its ability to predict unseen views, without requiring a ground-truth geometry model of the scene. This definition of reconstruction quality exactly matches McMillan and Bishop's [1995] definition of the plenoptic function and thus makes all IBMR representations directly comparable and therefore accessible for a general IBMR benchmark. The questions we try to answer in the following are: What have we achieved? When is virtual rephotography useful? When should it not be used and what are its limitations?

While the high-level idea of novel view prediction error has already been used for evaluation in other areas (*e.g.*, optical flow [Szeliski 1999]), we were the first to devise desiderata and meta-evaluation experiments which allowed us to put this framework to the test. Specifically, we demonstrated the fulfillment of the ordering criterion, which states that if representation A is a more accurate visual representation of the underlying real-world scene than representation B, A should receive a lower error than B. Based on our experiments, our key results are the following findings:

1. Virtual rephotography reports reconstruction quality degradation when we introduce artificial defects into a ground-truth model (Section 7.1),
2. it reports reconstruction quality degradation when we impair the input data of reconstruction algorithms (Section 7.2),
3. it is correlated to geometric error but also captures complementary aspects which are consistent with our visual intuition (Section 7.4),
4. and it ranks different IBMR representations in an order that is consistent with our visual intuition (Section 7.5), which is also supported by a correlation between human judgment and rephoto error (Section 7.6).

The ordering criterion and our experiments allowed us to vary the image difference metrics within the virtual rephotography framework and verify their suitability. The findings above hold for all investigated *patch-based metrics*: 1-NCC, Census, ZSSD, DSSIM, and iCID. This does not come as a complete surprise: In the context of detecting global illumination and rendering artifacts, Mantiuk [2013] "did not find evidence in [his] data set that any of the metrics [...] is significantly better than any other metric." We want to qualify this statement a bit and stress that 1-NCC and Census consistently performed best in our experiments: They separated the box plots in the experiments of Section 7.2 clearly and they correlated strongly with

the Middlebury error and user rankings. Further, the error projections of 1-NCC in conjunction with median aggregation are more compact than those of Census (see Figure 8.1) and most consistent with errors that we found while manually inspecting the reconstructions. Therefore, we give 1-NCC a slight preference over Census.

In an application such as reconstruction for 3D printing, which exclusively focuses on geometric accuracy, measuring Middlebury-style geometric accuracy most likely remains a key method[67] if we have a real-world object and corresponding ground truth available for testing purposes. Contrarily, if an application either focuses on aspects orthogonal to geometric accuracy such as texture quality, or if no high-precision ground-truth geometry[68] is available, virtual rephotography may be the method of choice. Based on our Findings 1–3 above, we can confidently say that virtual rephotography is able to detect geometric reconstruction errors solely from input images. As discussed earlier, the reason for this is that geometric errors typically manifest themselves as errors in the predicted novel views unless an erroneous region is looked at more or less from the front and is untextured. Not requiring geometric ground truth is our method's main benefit. It allows for many new and important use cases such as determining low-quality scene regions and guiding users into those regions to capture more images and improve the reconstruction. Especially when capturing and evaluating previously unknown scenes, one of course has to resort to methods that do not require ground-truth geometry.

Another question is to what extent virtual rephotography correlates with human judgment. In Figure 5.2, its correlation to human judgment seems to be stronger than that of geometric accuracy, but this example was handpicked to underline the necessity of measures complementing geometric accuracy. At the end of Section 7.6, we found that the correlation of rephoto error vs. humans is stronger than that of geometric error vs. humans. Even though this evaluation was only done on the relatively small and controlled Middlebury-5 dataset, we argue that judging the correctness of images is closer to how humans – including 3D reconstruction and computer graphics experts – evaluate the accuracy of 3D scenes: They are visual beings and cannot assess the geometric accuracy of 3D scenes directly, especially if no ground truth is available. They instead look at renderings. Arguably, their advantage over virtual rephotography is that they can freely translate and rotate the virtual scene in a viewer and compare it with their prior knowledge about plausible surfaces that they learned in the real world or from other reconstructions. Virtual rephotography has to make do with a handful of test images. Nevertheless, we demonstrated that virtual rephotography can capture aspects of reconstructions that a geometric measure cannot capture by design. In addition, rephoto errors are meaningful to humans when used for error localization on geometry (Section 8.1), *i.e.*, rephoto errors typically correspond with errors that humans find when manually

[67] One may prefer the similar measures from Knapitsch et al. [2017] that work on point clouds and do not require closed surfaces. But in any case, in applications with a primary focus on geometry and with available ground truth, geometric measures may be preferable over our measure.

[68] Ground truth typically needs to be much more accurate than the errors one tries to measure.

examining reconstructions. Thinking a bit into the future, a strong correlation with human judgment may be indispensable in, *e.g.*, 3D reconstructions for games or virtual/augmented reality. At the end of the day, models in visual applications have to appear accurate to the human eye and geometric inaccuracies may be irrelevant as long as they do not cause perceivable inaccuracies.

9.1. Limitations

Some important limitations of our approach are the following.

9.1.1. Pin-pointing Error Sources

Due to its black box view on image-based modeling and rendering, virtual rephotography can tell for a wide variety of reconstruction systems how well they perform on a specific scene. However, if a reconstruction contains an error, this black box also prevents pinpointing what step of a reconstruction pipeline caused this. *E.g.*, when evaluating an MVS and texturing pipeline which produces errors in the MVS step, we can most likely detect the existence of this error since geometric errors frequently produce visual errors and with error projection we may even be able to localize these geometric errors. But to precisely debug only the MVS step, one may have to resort to a geometric measure. Thus, virtual rephotography does not replace but instead complements other evaluation metrics that focus on specific pipeline steps or reconstruction representations.

Using a specialized evaluation method such as Middlebury or a general method such as virtual rephotography, is a trade-off: An evaluation framework can either be very general and enable comparability of many different methods/representations, or it can precisely detect certain error types and sources.

Knapitsch et al. [2017] encountered a very similar problem in their joint SfM and MVS benchmark. Similar to them, we could, *e.g.*, execute the Cartesian product of all MVS algorithms vs. all surface reconstruction algorithms in order to gain some insight into what building block may have caused a strong error response.

9.1.2. Realistic Light Transport

Another limitation revolves around various issues connected to realistic light transport: Currently we do not evaluate whether a system handles properties and effects such as surface albedo, BRDF, shading, interreflection, illumination, or camera response curves correctly. They can in principle be evaluated, but one would have to use image difference metrics that are less luminance-invariant than the investigated ones, *e.g.*, the mean squared error. Further, one would need to measure and provide all information about the test images that is necessary for correct novel-view prediction but cannot be inferred from the training images, *e.g.*, illumination, exposure time, camera response curve, or even shadows cast by objects that are not visible in any of the images. Acquiring ground-truth datasets, *e.g.*, for benchmarking would

Figure 9.1.: A person in one of the test images (*left*). Since he is missing in the reconstruction, he appears in the 1-NCC error image (*center*), but not in the error projection (*right*).

then become significantly more complicated and time-consuming. In certain settings it may be appropriate to incorporate the above effects, but given that those are currently not considered by most 3D reconstruction systems and that virtual rephotography already enables a large number of new applications when using the metrics we investigated, we believe that our choice of metrics is appropriate for the time being.

9.1.3. Test Data Quality

Unless a dataset has been captured under controlled lab conditions, it will most likely have imperfections such as under- or overexposure, blur outside the depth of field, or moving objects such as plants or pedestrians. While imperfections in the training data are acceptable in principle and can even be regarded as challenges that are necessary to advance the state of the art in image-based modeling, imperfections in the test data are clearly undesirable since it is impossible for IBMR algorithms to predict these imperfections correctly and achieve a perfect score. Our metrics measure differences between a rephoto and a test image and cannot tell which of both is "erroneous". If a test image contains, *e.g.*, an occluder such as the person in Figure 9.1 and the rephoto does not contain it because moving objects tend to not reconstruct, the difference will be high in that image region (see Figure 9.1, center) even though the reconstruction is in essence a good one.

Interestingly, such differences are often not visible in the error projections (Figure 9.1, right): If an occluder has not been reconstructed, the high difference is projected to the scene region behind it. Typically, scene parts are seen redundantly by many views and these views may or may not consistently see the occluder in front of the same background, depending on the occluder's and the camera's movement. If an occluder stays at the same spot and is close to the geometry behind it, all views will consistently project their error to the same region (Figure 9.2, left). If, on the other hand, the occluder moves or is far away from the background, the error may be projected to inconsistent places and may be averaged out (Figure 9.2, right).

Thus, virtual rephotography's error projections only detect errors in surface regions where projections from multiple test views agree on the error. We can see this

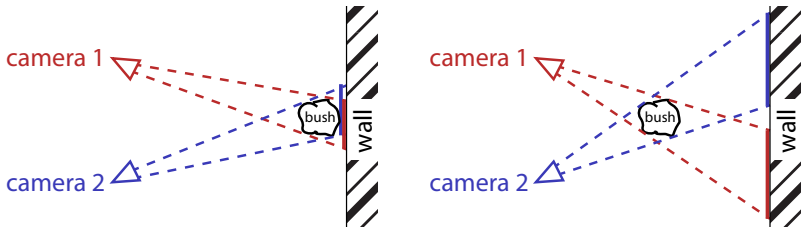


Figure 9.2.: If an occluder was not reconstructed, it produces a rephoto error in all observing cameras. *Left:* If the occluder is close to its background (*e.g.*, a wall), these errors project to more or less consistent regions (the solid red and blue lines on the wall). *Right:* If the occluder is far away from its background, the projections are inconsistent.

effect in practice in Figure 8.3: The pedestrians only show up in the error projections wherever they were baked into the texture and are thus present in the rephotos, but not where they are present in test images. Contrarily, the plant in Figure 8.2 does show up in the error projections because it is very close to the stone behind it. In our experience, the same effect occurs with other test data imperfections such as blur: For the purpose of error localization, imperfect test data are not a fundamental issue if there are enough test images to average them out.

Imperfect test data do, however, have an influence on global reconstruction quality scores such as those reported by our benchmark. When capturing our two datasets outside the lab (Mattia and the Deer), we therefore took care to avoid and alleviate imperfections as much as possible. Moving objects could further be ruled out by taking multiple photos from each viewpoint and using median images to detect dynamic image content that could then be excluded from the evaluation. Once enough independent benchmark submissions[69] exist, an alternative would be to analyze inter-submission agreement and agreement between submissions and test data: If multiple reconstruction algorithms independently lead to the same rephoto but disagree with the test image, the test image is likely "incorrect", *i.e.*, it contains blur, dynamic objects, *etc.*, and should be discarded from the evaluation in the future.[70]

9.2. Future Work

In the future, it would be worthwhile to incorporate visual accuracy in many aspects of image-based modeling and rendering. This has already been done, *e.g.*, by

[69] Here, (in)dependence means that a pipeline consisting of MVS algorithm A and texturing algorithm B might produce similar errors as an unstructured Lumigraph that uses algorithm A's depth maps as proxy. Independent submissions should not share algorithmic building blocks.

[70] Modifying or even generating "ground truth" based on submissions is not a completely unheard-of thing: In their "Benchmark on High Density Aerial Image Matching", Haala et al. [2014] wrote: "As a common reference surface, the results provided by the participants were used to generate a median [Digital Surface Model]. Of course this median does not provide an independent ground truth. However, it can be used very well to illustrate differences between the respective solutions and thus give hints to situations which are potentially difficult for image matching."

Vanhoey et al. [2015], who simplify meshes, and Morvan and O'Sullivan [2009], who remove images from unstructured Lumigraphs, respectively, such that the visual appearance of the final result is compromised as little as possible. This can even be taken further by not requiring the same visual quality in each part of the scene: Models do not need to be very accurate in regions where inaccuracies will never be uncovered by any test image. If, *e.g.*, a reconstruction will be used in a virtual reality game, potential player positions will all be close to the ground plane and it therefore suffices to only pick images close to the ground as test images. They may even be confined to a certain path along the ground. The training images do not necessarily have to be restricted in the same way.

Virtual rephotography could also be used in view planning for 3D reconstruction image acquisition: The works by Hoppe et al. [2012b], Mauro et al. [2014], and Roberts et al. [2017] first create an initial reconstruction from a completely uninformed image acquisition phase (with a drone flying at a safe altitude) and use this scene approximation for computing viewpoints from which new images should be taken in a second acquisition phase in order to improve the reconstruction quality. All these works plan new viewpoints only based on geometric criteria (viewing angle, ground sampling density, overlap with other images, *etc.*). Our localized error visualizations (Section 8.1) could help to guide the second capture phase into regions with low *visual* reconstruction quality. What helps us here again, is that virtual rephotography requires no ground truth and simply measures the consistency of reconstructions. Of course, virtual rephotography alone cannot distinguish between scene regions that are fundamentally unreconstructable and those that the reconstruction algorithm at hand simply failed to reconstruct properly even though it is in principle reconstructable with the current or near-future state of the art in 3D reconstruction, but it gives us at least an indication. Using an initial reconstruction approximation to find errors with virtual rephotography and then using those errors to decide where to move in the scene to take more pictures would close a loop in the sense of Soatto [2009] and Gibson [1979, page 223]:

> "[We] must perceive in order to move, but we must also move in order to perceive."

Another promising research direction is community photo collections, *i.e.*, tourist photos from photo sharing sites. In Section 3.5, we already pointed out that these are very problematic for texturing because their challenges are much stronger than, for example, in datasets captured by an expert within a short time frame (*i.e.*, no major illumination changes) with a good camera. Unsurprisingly, community photo collection datasets are also very challenging for virtual rephotography. Most problematic here are test image imperfections that we discussed above. In this case we even have to deal with night-time images or images where more than half of the picture is occluded by people (*e.g.*, in selfies). In contrast to, *e.g.*, benchmark datasets, in this setting we cannot eliminate challenges by capturing images carefully, but we have to work with the data that we have. As already pointed out,

Figure 9.3.: Detail of a reconstruction (*left*) and its 1-NCC error projection (*right*) of Notre Dame based on 699 Flickr images.

our error projection method is partially robust towards such effects because it averages out challenging test data. In our experience, this seems to be sufficient for some community photo collection datasets and the errors highlighted by the 1-NCC projection correspond with errors we found during manual inspection of the reconstructions: *E.g.*, in Figure 9.3, the 1-NCC projection detects dark, blurry texture from a night-time image on the model's left side and heavily distorted texture on the model's right. For other datasets, however, it clearly does not work and the error projections correspond much less with what we found to be erroneous. In the future, we would therefore like to investigate means to make our evaluation scheme more robust with respect to such challenging datasets.

Like many previous works [Snavely et al. 2006, Agarwal et al. 2009, Frahm et al. 2010, Heinly et al. 2015, Goesele et al. 2007, Furukawa et al. 2010, Schönberger et al. 2016] we are convinced that the internet with its never-ending supply of images is an immensely valuable data source. With internet data, it is for example possible to reconstruct and visualize scenes that have never been scanned with an active scanning method and that can no longer be scanned because they have been destroyed. Scenes such as the ancient city of Palmyra or the Great Buddhas of Bamiyan[71] could be reconstructed *and the reconstruction evaluated*, if enough pictures of it can be found on the internet. Even though the internet contains a lot of unsuitable and very noisy data, the information exists with such great redundancy that this may help to detect erroneous and noisy data. The great amount of data is a challenge and an opportunity: If we can make our algorithms robust enough to work fully automatically with data that users captured simply for their own and others' enjoyment and not with image-based modeling and rendering in mind, then we can work with any data.

[71] Grün et al. [2004] did a reconstruction of the Buddhas but based on analog images and manual correspondence annotations.

(Co-)Authored Publications

Waechter, M., Beljan, M., Fuhrmann, S., Moehrle, N., Kopf, J., and Goesele, M. (2017). Virtual rephotography: Novel view prediction error for 3D reconstruction. *ACM Transactions on Graphics*, 36(1).

Weber, N., **Waechter, M.**, Amend, S. C., Guthe, S., and Goesele, M. (2016). Rapid, detail-preserving image downscaling. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia)*, 35(6).

Thuerck, D., **Waechter, M.**, Widmer, S., von Buelow, M., Seemann, P., Pfetsch, M. E., and Goesele, M. (2016). A fast, massively parallel solver for large, irregular pairwise Markov random fields. In *High-Performance Graphics (HPG)*.

Fuhrmann, S., Langguth, F., Moehrle, N., **Waechter, M.**, and Goesele, M. (2015). MVE – An image-based reconstruction environment. *Computers & Graphics*, 53, Part A.

Waechter, M., Jaeger, K., Thuerck, D., Weissgraeber, S., Widmer, S., Goesele, M., and Hamacher, K. (2014a). Using graphics processing units to investigate molecular coevolution. *Concurrency and Computation: Practice and Experience*, 26(6).

Waechter, M., Moehrle, N., and Goesele, M. (2014b). Let there be color! Large-scale texturing of 3D reconstructions. *Springer Lecture Notes in Computer Science (Proceedings of the European Conference on Computer Vision)*, 8693.

Waechter, M., Jaeger, K., Weissgraeber, S., Widmer, S., Goesele, M., and Hamacher, K. (2012). Information-theoretic analysis of molecular (co)evolution using graphics processing units. In *HPDC Workshop on Emerging Computational Methods for the Life Sciences*.

Waechter, M., Hamacher, K., Hoffgaard, F., Widmer, S., and Goesele, M. (2011). Is your permutation algorithm unbiased for $n \neq 2^m$? *Springer Lecture Notes in Computer Science (Proceedings of the International Conference on Parallel Processing and Applied Mathematics)*, 7203.

Bibliography

Aanæs, H., Jensen, R. R., Vogiatzis, G., Tola, E., and Dahl, A. B. (2016). Large-scale data for multiple-view stereopsis. *International Journal of Computer Vision (IJCV)*, 120(2).

Adelson, E. H. and Bergen, J. R. (1991). The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*.

Adelson, E. H. and Wang, J. Y. A. (1992). Single lens stereo with a plenoptic camera. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2).

Agarwal, S., Snavely, N., Simon, I., Seitz, S. M., and Szeliski, R. (2009). Building Rome in a day. In *International Conference on Computer Vision (ICCV)*.

Allène, C., Pons, J.-P., and Keriven, R. (2008). Seamless image-based texture atlases using multi-band blending. In *International Conference on Pattern Recognition (ICPR)*.

Avidan, S. and Shashua, A. (1997). Novel view synthesis in tensor space. In *Computer Vision and Pattern Recognition (CVPR)*.

Aydın, T. O., Mantiuk, R., Myszkowski, K., and Seidel, H.-P. (2008). Dynamic range independent image quality assessment. In *SIGGRAPH*.

Bae, S., Agarwala, A., and Durand, F. (2010). Computational rephotography. *Transactions on Graphics (TOG)*, 29(3).

Bagon, S. and Galun, M. (2012). A multiscale framework for challenging discrete optimization. In *NIPS Workshop on Optimization for Machine Learning (OPT)*.

Baker, S., Scharstein, D., Lewis, J. P., Roth, S., Black, M. J., and Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision (IJCV)*, 92(1).

Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*.

Berger, K., Lipski, C., Linz, C., Sellent, A., and Magnor, M. (2010). A ghosting artifact detector for interpolated image quality assessment. In *International Symposium on Consumer Electronics*.

Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, 48(3).

Blake, A., Kohli, P., and Rother, C. (2011). *Markov random fields for vision and image processing*. MIT Press.

Bleyer, M., Rhemann, C., and Rother, C. (2011). PatchMatch stereo – Stereo matching with slanted support windows. In *British Machine Vision Conference (BMVC)*.

Boukhayma, A., Tsiminaki, V., Franco, J.-S., and Boyer, E. (2016). Eigen appearance maps of dynamic shapes. In *European Conference on Computer Vision (ECCV)*.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(11).

Buehler, C., Bosse, M., McMillan, L., Gortler, S., and Cohen, M. F. (2001). Unstructured Lumigraph rendering. In *SIGGRAPH*.

Calakli, F., Ulusoy, A. O., Restrepo, M. I., Taubin, G., and Mundy, J. L. (2012). High resolution surface reconstruction from multi-view aerial imagery. In *3DIMPVT*.

Callieri, M., Cignoni, P., Corsini, M., and Scopigno, R. (2008). Masked photo blending: Mapping dense photographic dataset on high-resolution sampled 3D models. *Computers & Graphics*, 32(4).

Campbell, N. D., Vogiatzis, G., Hernández, C., and Cipolla, R. (2008). Using multiple hypotheses to improve depth-maps for multi-view stereo. In *European Conference on Computer Vision (ECCV)*.

Chai, J.-X., Tong, X., Chan, S.-C., and Shum, H.-Y. (2000). Plenoptic sampling. In *SIGGRAPH*.

Chen, Q. and Koltun, V. (2014). Fast MRF optimization with application to depth reconstruction. In *Computer Vision and Pattern Recognition (CVPR)*.

Chen, S. E. and Williams, L. (1993). View interpolation for image synthesis. In *SIGGRAPH*.

Chen, Z., Zhou, J., Chen, Y., and Wang, G. (2012). 3D texture mapping in multiview reconstruction. In *International Symposium on Advances in Visual Computing (ISVC)*.

Crandall, D. J., Owens, A., Snavely, N., and Huttenlocher, D. P. (2013). SfM with MRFs: Discrete-continuous optimization for large-scale structure from motion. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(12).

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *SIGGRAPH*.

Daly, S. (1993). The visible differences predictor: An algorithm for the assessment of image fidelity. In *Digital Images and Human Vision*. MIT Press.

Debevec, P. E., Taylor, C. J., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH*.

Dellepiane, M., Marroquim, R., Callieri, M., Cignoni, P., and Scopigno, R. (2012). Flow-based local optimization for image-to-geometry projection. *Transactions on Visualization and Computer Graphics (TVCG)*, 18(3).

Delong, A., Osokin, A., Isack, H. N., and Boykov, Y. (2012). Fast approximate energy minimization with label costs. *International Journal of Computer Vision (IJCV)*, 96(1).

Devaux, A. and Brédif, M. (2016). Realtime projective multi-texturing of pointclouds and meshes for a realistic street-view web navigation. In *International Conference on 3D Web Technology (Web3D)*.

Djurdjani and Laksono, D. (2016). Open source stack for structure from motion 3D reconstruction: A geometric overview. In *International Annual Engineering Seminar (InAES)*.

Eisemann, M., De Decker, B., Magnor, M., Bekaert, P., de Aguiar, E., Ahmed, N., Theobalt, C., and Sellent, A. (2008). Floating textures. *Eurographics (EG)*, 27(2).

Faugeras, O. D. (1992). What can be seen in three dimensions with an uncalibrated stereo rig? In *European Conference on Computer Vision (ECCV)*.

Felzenszwalb, P. and Huttenlocher, D. (2006). Efficient belief propagation for early vision. *International Journal of Computer Vision (IJCV)*, 70(1).

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6).

Fitzgibbon, A., Wexler, Y., and Zisserman, A. (2005). Image-based rendering using image-based priors. *International Journal of Computer Vision (IJCV)*, 63(2).

Förstner, W. (1996). 10 pros and cons against performance characterization of vision algorithms. In *Workshop on Performance Characteristics of Vision Algorithms*.

Frahm, J.-M., Fite-Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.-H., Dunn, E., Clipp, B., Lazebnik, S., and Pollefeys, M. (2010). Building Rome on a cloudless day. In *European Conference on Computer Vision (ECCV)*.

Fuhrmann, S. and Goesele, M. (2011). Fusion of depth maps with multiple scales. In *SIGGRAPH Asia*.

Fuhrmann, S. and Goesele, M. (2014). Floating scale surface reconstruction. In *SIGGRAPH*.

Fuhrmann, S., Langguth, F., and Goesele, M. (2014). MVE – A multi-view reconstruction environment. In *Eurographics Workshop on Graphics and Cultural Heritage*.

Fuhrmann, S., Langguth, F., Moehrle, N., Waechter, M., and Goesele, M. (2015). MVE – An image-based reconstruction environment. *Computers & Graphics*, 53, Part A.

Furukawa, Y., Curless, B., Seitz, S. M., and Szeliski, R. (2010). Towards internet-scale multi-view stereo. In *Computer Vision and Pattern Recognition (CVPR)*.

Furukawa, Y. and Ponce, J. (2010). Accurate, dense, and robust multi-view stereopsis. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(8).

Gal, R., Wexler, Y., Ofek, E., Hoppe, H., and Cohen-Or, D. (2010). Seamless montage for texturing models. *Computer Graphics Forum*, 29(2).

Garcia-Dorado, I., Demir, I., and Aliaga, D. G. (2013). Automatic urban modeling using volumetric reconstruction with surface graph cuts. *Computers & Graphics*, 37(7).

Gibson, J. J. (1979). *The ecological approach to visual perception*. Lawrence Erlbaum Associates, Inc.

Goesele, M., Snavely, N., Curless, B., Hoppe, H., and Seitz, S. M. (2007). Multi-view stereo for community photo collections. In *International Conference on Computer Vision (ICCV)*.

Goldluecke, B., Aubry, M., Kolev, K., and Cremers, D. (2014). A super-resolution framework for high-accuracy multiview reconstruction. *International Journal of Computer Vision (IJCV)*, 106(2).

Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. (1996). The Lumigraph. In *SIGGRAPH*.

Grammatikopoulos, L., Kalisperakis, I., Karras, G., and Petsa, E. (2007). Automatic multi-view texture mapping of 3D surface projections. In *3D Virtual Reconstruction and Visualization of Complex Architectures (3D Arch)*.

Gross, C. G. (1999). The fire that comes from the eye. *The Neuroscientist*, 5(1).

Grün, A., Remondino, F., and Zhang, L. (2004). Photogrammetric reconstruction of the Great Buddha of Bamiyan, Afghanistan. *The Photogrammetric Record*, 19(107).

Guennebaud, G., Jacob, B., and others (2010). Eigen v3. `http://eigen.tuxfamily.org`.

Guthe, S., Cunningham, D. W., Schardt, P., and Goesele, M. (2016). Ghosting and popping detection for image-based rendering. In *3DTV Conference*.

Haala, N. et al. (2014). Benchmark on high density aerial image matching. `www.ifp.uni-stuttgart.de/ISPRS-EuroSDR/ImageMatching`.

Haber, T., Fuchs, C., Bekaert, P., Seidel, H.-P., Goesele, M., and Lensch, H. P. A. (2009). Relighting objects from image collections. In *Computer Vision and Pattern Recognition (CVPR)*.

Hafeez, J., Hamacher, A., Son, H., Pardeshi, S., and Lee, S.-H. (2016). Workflow evaluation for optimized image-based 3D model reconstruction. In *International Conference on Electronics, Electrical Engineering, Computer Science: Innovation and Convergence*.

Handa, A., Patraucean, V., Stent, S., and Cipolla, R. (2016). SceneNet: An annotated model generator for indoor scene understanding. In *International Conference on Robotics and Automation*.

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Alvey Vision Conference*.

Hartley, R. I. (1997). In defense of the eight-point algorithm. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 19(6).

Hartley, R. I. and Zisserman, A. (2004). *Multiple view geometry in computer vision*. Cambridge University Press, 2nd edition.

Havlena, M. and Schindler, K. (2014). VocMatch: Efficient multiview correspondence for structure from motion. In *European Conference on Computer Vision (ECCV)*.

Heindl, C., Akkaladevi, S. C., and Bauer, H. (2016a). Capturing photorealistic and printable 3D models using low-cost hardware. In *International Symposium on Advances in Visual Computing (ISVC)*.

Heindl, C., Akkaladevi, S. C., and Bauer, H. (2016b). Photorealistic texturing of human busts reconstructions. In *3D Body Scanning Technologies*.

Heinly, J., Schönberger, J. L., Dunn, E., and Frahm, J.-M. (2015). Reconstructing the world* in six days *(as captured by the Yahoo 100 million image dataset). In *Computer Vision and Pattern Recognition (CVPR)*.

Hernández, J. D., Istenic, K., Gracias, N., García, R., Ridao, P., and Carreras, M. (2016a). Autonomous seabed inspection for environmental monitoring. In *Robot 2015: Second Iberian Robotics Conference: Advances in Robotics*.

Hernández, J. D., Istenic, K., Gracias, N., Palomeras, N., Campos, R., Vidal, E., García, R., and Carreras, M. (2016b). Autonomous underwater navigation and optical mapping in unknown natural environments. *Sensors*, 16(8).

Hirschmüller, H. (2008). Stereo processing by semiglobal matching and mutual information. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(2).

Hirschmüller, H. and Scharstein, D. (2009). Evaluation of stereo matching costs on images with radiometric differences. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 31(9).

Hofsetz, C., Ng, K., Chen, G., McGuinness, P., Max, N., and Liu, Y. (2004). Image-based rendering of range data with estimated depth uncertainty. *Computer Graphics and Applications*, 24(4).

Holzmann, T., Fraundorfer, F., and Bischof, H. (2016). Regularized 3D modeling from noisy building reconstructions. In *International Conference on 3D Vision (3DV)*.

Hongxing, Y., Li, G., Li, Y., and Long, C. (2010). Multi-view reconstruction using band graph-cuts. *Journal of Computer-Aided Design & Computer Graphics*, 4.

Hoppe, C., Klopschitz, M., Rumpler, M., Wendel, A., Kluckner, S., Bischof, H., and Reitmayr, G. (2012a). Online feedback for structure-from-motion image acquisition. In *British Machine Vision Conference (BMVC)*.

Hoppe, C., Wendel, A., Zollmann, S., Pirker, K., Irschara, A., Bischof, H., and Kluckner, S. (2012b). Photogrammetric camera network design for micro aerial vehicles. In *Computer Vision Winter Workshop*.

Jamriska, O., Sýkora, D., and Hornung, A. (2012). Cache-efficient graph cuts on structured grids. In *Computer Vision and Pattern Recognition (CVPR)*.

Jancosek, M. and Pajdla, T. (2011). Multi-view reconstruction preserving weakly-supported surfaces. In *Computer Vision and Pattern Recognition (CVPR)*.

Kang, S. B., Li, Y., Tong, X., and Shum, H.-Y. (2006). Image-based rendering. *Foundations and Trends in Computer Graphics and Vision*, 2(3).

Kang, S. B. and Szeliski, R. (2004). Extracting view-dependent depth maps from a collection of images. *International Journal of Computer Vision (IJCV)*, 58(2).

Kang, S. B., Szeliski, R., and Anandan, P. (2000). The geometry-image representation tradeoff for rendering. In *International Conference on Image Processing (ICIP)*.

Kappes, J. H., Andres, B., Hamprecht, F. A., Schnörr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B. X., Kröger, T., Lellmann, J., Komodakis, N., Savchynskyy, B., and Rother, C. (2015). A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision (IJCV)*, 115(2).

Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing (SGP)*.

Kelm, B. M., Mueller, N., Menze, B. H., and Hamprecht, F. A. (2006). Bayesian estimation of smooth parameter maps for dynamic contrast-enhanced MR images with block-ICM. In *CVPR Workshop on Mathematical Methods in Biomedical Image Analysis*.

Kim, T., Nowozin, S., Kohli, P., and Yoo, C. D. (2011). Variable grouping for energy minimization. In *Computer Vision and Pattern Recognition (CVPR)*.

Knapitsch, A., Park, J., Zhou, Q.-Y., and Koltun, V. (2017). Tanks and temples: Benchmarking large-scale scene reconstruction. In *SIGGRAPH*.

Kneip, L., Scaramuzza, D., and Siegwart, R. (2011). A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *Computer Vision and Pattern Recognition (CVPR)*.

Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(10).

Kolmogorov, V. and Zabih, R. (2002). Multi-camera scene reconstruction via graph cuts. In *European Conference on Computer Vision (ECCV)*.

Komodakis, N. and Tziritas, G. (2007). Approximate labeling via graph cuts based on linear programming. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(8).

Kopf, J., Cohen, M. F., and Szeliski, R. (2014). First-person hyper-lapse videos. In *SIGGRAPH*.

Kopf, J., Langguth, F., Scharstein, D., Szeliski, R., and Goesele, M. (2013). Image-based rendering in the gradient domain. In *SIGGRAPH Asia*.

Krispel, U., Evers, H. L., Tamke, M., Viehauser, R., and Fellner, D. W. (2015). Automatic texture and orthophoto generation from registered panoramic views. *ISPRS International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*.

Kutulakos, K. N. and Seitz, S. M. (2000). A theory of shape by space carving. *International Journal of Computer Vision (IJCV)*, 38(3).

Labrie-Larrivée, F., Laurendeau, D., and Lalonde, J. F. (2016). Depth texture synthesis for realistic architectural modeling. In *Computer and Robot Vision*.

Lambert, J. H. and Anding, E. (1892). *Photometria, sive de mensura et gradibus luminus, colorum et umbrae*. Engelmann, abridged German translation edition.

Laveau, S. and Faugeras, O. D. (1994). 3-D scene representation as a collection of images. In *International Conference on Pattern Recognition (ICPR)*.

Leclerc, Y. G., Luong, Q.-T., and Fua, P. (2000). Measuring the self-consistency of stereo algorithms. In *European Conference on Computer Vision (ECCV)*.

Lempitsky, V., Rother, C., Roth, S., and Blake, A. (2010). Fusion moves for Markov random field optimization. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(8).

Lempitsky, V. S. and Ivanov, D. V. (2007). Seamless mosaicing of image-based texture maps. In *Computer Vision and Pattern Recognition (CVPR)*.

Lengyel, J. (1998). The convergence of graphics and vision. *Computer*, 31(7).

Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *SIGGRAPH*.

Ley, A. and Hellwich, O. (2016). Depth map based facade abstraction from noisy multi-view stereo point clouds. In *German Conference on Pattern Recognition (GCPR)*.

Lhuillier, M. and Quan, L. (1999). Image interpolation by joint view triangulation. In *Computer Vision and Pattern Recognition (CVPR)*.

Lhuillier, M. and Quan, L. (2003). Image-based rendering by joint view triangulation. *Transactions on Circuits and Systems for Video Technology*, 13(11).

Li, C., Zhou, L., and Chen, W. (2016). Automatic pose estimation of uncalibrated multi-view images based on a planar object with a predefined contour model. *ISPRS International Journal of Geo-Information*, 5(12).

Lin, W. Y., Wang, F., Cheng, M. M., Yeung, S. K., Torr, P. H. S., Do, M. N., and Lu, J. (2017). CODE: Coherence based decision boundaries for feature correspondence. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

Liu, J., Liu, B., Fang, T., Huo, H., and Zhao, Y. (2016). Seamless texture mapping algorithm for image-based three-dimensional reconstruction. *Journal of Electronic Imaging*, 25(5).

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *International Conference on Computer Vision (ICCV)*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2).

Lurie, K. L., Angst, R., Zlatev, D. V., Liao, J. C., and Ellerbee Bowden, A. K. (2017). 3D reconstruction of cystoscopy videos for comprehensive bladder records. *Biomedical Optics Express*, 8(4).

Maier, R., Stückler, J., and Cremers, D. (2015). Super-resolution keyframe fusion for 3D modeling with high-quality textures. In *International Conference on 3D Vision (3DV)*.

Mantiuk, R. (2013). Quantifying image quality in graphics: Perspective on subjective and objective metrics and their performance. In *Human Vision and Electronic Imaging*.

Mantiuk, R., Kim, K. J., Rempel, A. G., and Heidrich, W. (2011). HDR-VDP-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions. In *SIGGRAPH*.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *International Conference on Computer Vision (ICCV)*.

Matusik, W., Buehler, C., Raskar, R., Gortler, S. J., and McMillan, L. (2000). Image-based visual hulls. In *SIGGRAPH*.

Mauro, M., Riemenschneider, H., Signoroni, A., Leonardi, R., and Van Gool, L. (2014). A unified framework for content-aware view selection and planning through view importance. In *British Machine Vision Conference (BMVC)*.

McMillan, L. and Bishop, G. (1995). Plenoptic modeling: An image-based rendering system. In *SIGGRAPH*.

Merrell, P., Akbarzadeh, A., Wang, L., Mordohai, P., Frahm, J.-M., Yang, R., Nistér, D., and Pollefeys, M. (2007a). Real-time visibility-based fusion of depth maps. In *International Conference on Computer Vision (ICCV)*.

Merrell, P., Mordohai, P., Frahm, J.-M., and Pollefeys, M. (2007b). Evaluation of large scale scene reconstruction. In *ICCV Workshop on Visual Representations and Modeling of Large-scale Environments (VRML)*.

Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 27(10).

Miksik, O. and Mikolajczyk, K. (2012). Evaluation of local detectors and descriptors for fast feature matching. In *International Conference on Pattern Recognition (ICPR)*.

Morvan, Y. and O'Sullivan, C. (2009). A perceptual approach to trimming and tuning unstructured Lumigraphs. *Transactions on Applied Perception (TAP)*, 5(4).

Mücke, P., Klowsky, R., and Goesele, M. (2011). Surface reconstruction from multi-resolution sample points. In *Vision, Modeling, and Visualization (VMV)*.

Mueller, K., Zabulis, X., Smolic, A., and Wiegand, T. (2005). Evaluation of 3D reconstruction using multiview backprojection. In *Workshop on Immersive Communication and Broadcast Systems*.

Mustafa, M., Guthe, S., and Magnor, M. (2012). Single trial EEG classification of artifacts in videos. *Transactions on Applied Perception (TAP)*, 9(3).

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohi, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). KinectFusion: Real-time dense surface mapping and tracking. In *International Symposium on Mixed and Augmented Reality (ISMAR)*.

Ng, R., Levoy, M., Brédif, M., Duval, G., Horowitz, M., and Hanrahan, P. (2005). Light field photography with a hand-held plenoptic camera. Technical report, Stanford University.

Nistér, D. and Stewénius, H. (2006). Scalable recognition with a vocabulary tree. In *Computer Vision and Pattern Recognition (CVPR)*.

Oliveira, M. M. (2002). Image-based modeling and rendering techniques: A survey. *Revista de Informática Teórica e Aplicada (RITA)*, 9(2).

Pagés, R., Berjón, D., Morán, F., and García, N. (2015). Seamless, static multi-texturing of 3D meshes. *Computer Graphics Forum*, 34(1).

Pagés, R., Fuentes, D., and Morán, F. (2011). ITEM: Inter-texture error measurement for 3D meshes. In *International Conference on 3D Web Technology (Web3D)*.

Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. *Transactions on Graphics (TOG)*, 22(3).

Perlin, K. (2002). Improving noise. In *SIGGRAPH*.

Plötz, T. and Roth, S. (2017). Automatic registration of images to untextured geometry using average shading gradients. *International Journal of Computer Vision (IJCV)*.

Pont-Tuset, J. and Marques, F. (2013). Measures and meta-measures for the supervised evaluation of image segmentation. In *Computer Vision and Pattern Recognition (CVPR)*.

Pont-Tuset, J. and Marques, F. (2016). Supervised evaluation of image segmentation and object proposal techniques. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 38(7).

Potmesil, M. (1987). Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1).

Preiss, J., Fernandes, F., and Urban, P. (2014). Color-image quality assessment: From prediction to optimization. *Transactions on Image Processing (TIP)*, 23(3).

Ramanarayanan, G., Ferwerda, J., Walter, B., and Bala, K. (2007). Visual equivalence: Towards a new standard for image fidelity. In *SIGGRAPH*.

Roberts, M., Dey, D., Truong, A., Sinha, S., Shah, S., Kapoor, A., Hanrahan, P., and Joshi, N. (2017). Submodular trajectory optimization for aerial 3D scanning. `https://arxiv.org/abs/1705.00703v2`.

Rothermel, M. (2016). *Development of a SGM-Based Multi-View Reconstruction Framework for Aerial Imagery*. PhD thesis, Universität Stuttgart.

Rumpler, M., Tscharf, A., Mostegel, C., Daftry, S., Hoppe, C., Prettenthaler, R., Fraundorfer, F., Mayer, G., and Bischof, H. (2017). Evaluations on multi-scale camera networks for precise and geo-accurate reconstructions from aerial and terrestrial images with user guidance. *Computer Vision and Image Understanding (CVIU)*, 157.

Scharstein, D. (1999). *View synthesis using stereo vision*. Springer-Verlag.

Scharstein, D. and Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition (CVPR)*.

Schönberger, J. L., Zheng, E., Pollefeys, M., and Frahm, J.-M. (2016). Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*.

Schöps, T., Schönberger, J. L., Galliani, S., Sattler, T., Schindler, K., Pollefeys, M., and Geiger, A. (2017). A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *Computer Vision and Pattern Recognition (CVPR)*.

Schwarz, M. and Stamminger, M. (2009). On predicting visual popping in dynamic scenes. In *Applied Perception in Graphics and Visualization*.

Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer Vision and Pattern Recognition (CVPR)*.

Seitz, S. M. and Dyer, C. R. (1996). View morphing. In *SIGGRAPH*.

Shade, J., Gortler, S., He, L.-w., and Szeliski, R. (1998). Layered depth images. In *SIGGRAPH*.

Shan, Q., Adams, R., Curless, B., Furukawa, Y., and Seitz, S. M. (2013). The visual Turing test for scene reconstruction. In *International Conference on 3D Vision (3DV)*.

Shan, Q., Curless, B., Furukawa, Y., Hernandez, C., and Seitz, S. M. (2014a). Occluding contours for multi-view stereo. In *Computer Vision and Pattern Recognition (CVPR)*.

Shan, Q., Curless, B., Furukawa, Y., Hernandez, C., and Seitz, S. M. (2014b). Photo uncrop. In *European Conference on Computer Vision (ECCV)*.

Sheffer, A., Praun, E., and Rose, K. (2006). Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision*, 2(2).

Shekhovtsov, A. and Hlavác, V. (2013). A distributed mincut/maxflow algorithm combining path augmentation and push-relabel. *International Journal of Computer Vision (IJCV)*, 104(3).

Shen, T., Wang, J., Fang, T., Zhu, S., and Quan, L. (2016). Color correction for image-based modeling in the large. In *Asian Conference on Computer Vision (ACCV)*.

Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University.

Shimony, S. E. (1994). Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68(2).

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR)*.

Shum, H. and Kang, S. B. (2000). A review of image-based rendering techniques. In *Visual Communications and Image Processing*.

Shum, H.-Y. and He, L.-W. (1999). Rendering with concentric mosaics. In *SIGGRAPH*.

Shum, H.-Y. and Szeliski, R. (1998). Construction and refinement of panoramic mosaics with global and local alignment. In *International Conference on Computer Vision (ICCV)*.

Sinha, S. N., Scharstein, D., and Szeliski, R. (2014). Efficient high-resolution stereo matching using local plane sweeps. In *Computer Vision and Pattern Recognition (CVPR)*.

Sinha, S. N., Steedly, D., Szeliski, R., Agrawala, M., and Pollefeys, M. (2008). Interactive 3D architectural modeling from unordered photo collections. In *SIGGRAPH Asia*.

Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo tourism: Exploring photo collections in 3D. In *SIGGRAPH*.

Soatto, S. (2009). Why I am interested in vision. `http://stefano.soatto.sity.com/ucla-computer-science-department`.

Strecha, C., von Hansen, W., Van Gool, L., Fua, P., and Thoennessen, U. (2008). On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *Computer Vision and Pattern Recognition (CVPR)*.

Szeliski, R. (1999). Prediction error as a quality metric for motion and stereo. In *International Conference on Computer Vision (ICCV)*.

Szeliski, R. (2010). *Computer vision: Algorithms and applications*. Springer-Verlag New York, Inc., 1st edition.

Szeliski, R. and Shum, H.-Y. (1997). Creating full view panoramic image mosaics and environment maps. In *SIGGRAPH*.

Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., and Rother, C. (2008). A comparative study of energy minimization methods for Markov random fields with smoothness-based priors. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(6).

Tan, Y., Kwoh, L., and Ong, S. (2008). Large scale texture mapping of building facades. *ISPRS International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37.

Thuerck, D., Waechter, M., Widmer, S., von Buelow, M., Seemann, P., Pfetsch, M. E., and Goesele, M. (2016). A fast, massively parallel solver for large, irregular pairwise Markov random fields. In *High-Performance Graphics (HPG)*.

Tompkin, J., Kim, M. H., Kim, K. I., Kautz, J., and Theobalt, C. (2013). Preference and artifact analysis for video transitions of places. *Transactions on Applied Perception (TAP)*, 10(3).

Tran, S. and Davis, L. (2006). 3D surface reconstruction using graph cuts with surface constraints. In *European Conference on Computer Vision (ECCV)*.

Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (1999). Bundle adjustment – A modern synthesis. In *International Workshop on Vision Algorithms*. Springer.

Tscharf, A., Rumpler, M., Fraundorfer, F., Mayer, G., and Bischof, H. (2015). On the use of UAVs in mining and archaeology – Geo-accurate 3D reconstructions using various platforms and terrestrial views. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*.

Tuite, K., Snavely, N., Hsiao, D.-Y., Tabing, N., and Popović, Z. (2011). PhotoCity: Training experts at large-scale image acquisition through a competitive game. In *SIGCHI*.

Ummenhofer, B. and Brox, T. (2013). Point-based 3D reconstruction of thin objects. In *International Conference on Computer Vision (ICCV)*.

Utomo, A. P. and Wibowo, C. P. (2017). 3D reconstruction of temples in the special region of Yogyakarta by using close-range photogrammetry. In *Seminar Nasional Teknologi Informasi dan Multimedia*.

Uyttendaele, M., Eden, A., and Skeliski, R. (2001). Eliminating ghosting and exposure artifacts in image mosaics. In *Computer Vision and Pattern Recognition (CVPR)*.

Vangorp, P., Chaurasia, G., Laffont, P.-Y., Fleming, R., and Drettakis, G. (2011). Perception of visual artifacts in image-based rendering of façades. In *Eurographics Symposium on Rendering (EGSR)*.

Vangorp, P., Richardt, C., Cooper, E. A., Chaurasia, G., Banks, M. S., and Drettakis, G. (2013). Perception of perspective distortions in image-based rendering. In *SIGGRAPH*.

Vanhoey, K., Sauvage, B., Kraemer, P., Larue, F., and Dischler, J.-M. (2015). Simplification of meshes with digitized radiance. *The Visual Computer*, 31(6–8).

Veksler, O. (2005). Stereo correspondence by dynamic programming on a tree. In *Computer Vision and Pattern Recognition (CVPR)*.

Velho, L. and Sossai Jr., J. (2007). Projective texture atlas construction for 3D photography. *The Visual Computer*, 23(9–11).

Waechter, M., Beljan, M., Fuhrmann, S., Moehrle, N., Kopf, J., and Goesele, M. (2017). Virtual rephotography: Novel view prediction error for 3D reconstruction. *Transactions on Graphics (TOG)*, 36(1).

Waechter, M., Hamacher, K., Hoffgaard, F., Widmer, S., and Goesele, M. (2011). Is your permutation algorithm unbiased for $n \neq 2^m$? In *Parallel Processing and Applied Mathematics (PPAM)*.

Waechter, M., Jaeger, K., Thuerck, D., Weissgraeber, S., Widmer, S., Goesele, M., and Hamacher, K. (2014a). Using graphics processing units to investigate molecular coevolution. *Concurrency and Computation: Practice and Experience*, 26(6).

Waechter, M., Jaeger, K., Weissgraeber, S., Widmer, S., Goesele, M., and Hamacher, K. (2012). Information-theoretic analysis of molecular (co)evolution using graphics processing units. In *HPDC Workshop on Emerging Computational Methods for the Life Sciences*.

Waechter, M., Moehrle, N., and Goesele, M. (2014b). Let there be color! Large-scale texturing of 3D reconstructions. In *European Conference on Computer Vision (ECCV)*.

Wang, L., Kang, S. B., Szeliski, R., and Shum, H.-Y. (2001). Optimal texture map reconstruction from multiple views. In *Computer Vision and Pattern Recognition (CVPR)*.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *Transactions on Image Processing (TIP)*, 13(4).

Wang, Z. and Geng, W. (2017). Generation of view-dependent textures for an inaccurate model. In *International Conference on Computer and Information Science (ICIS)*.

Webb, R. H., Boyer, D. E., and Turner, R. M. (2010). *Repeat photography*. Island Press.

Weber, N., Waechter, M., Amend, S. C., Guthe, S., and Goesele, M. (2016). Rapid, detail-preserving image downscaling. In *SIGGRAPH Asia*.

Weigel, C. and Fan, F. (2008). GPU-based 3D video object synthesis and its quality assessment. In *3DTV Conference*.

Weiss, Y. and Freeman, W. T. (2001). On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *Transactions on Information Theory (TIT)*, 47(2).

Winer, G. A., Cottrell, J. E., Gregg, V., Fournier, J. S., and Bica, L. A. (2002). Fundamentally misunderstanding visual perception. Adults' belief in visual emissions. *American Psychologist*, 57(6).

Yu, Y., Debevec, P., Malik, J., and Hawkins, T. (1999). Inverse global illumination: Recovering reflectance models of real scenes from photographs. In *SIGGRAPH*.

Zabih, R. and Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence. In *European Conference on Computer Vision (ECCV)*.

Zaharescu, A., Boyer, E., and Horaud, R. (2007). TransforMesh: A topology-adaptive mesh-based approach to surface evolution. In *Asian Conference on Computer Vision (ACCV)*.

Zitnick, C. L., Kang, S. B., Uyttendaele, M., Winder, S., and Szeliski, R. (2004). High-quality video view interpolation using a layered representation. In *SIGGRAPH*.

Zwicker, M., Pfister, H., van Baar, J., and Gross, M. (2001). Surface splatting. In *SIGGRAPH*.

Ehrenwörtliche Erklärung[72]

Hiermit erkläre ich, die vorgelegte Arbeit zur Erlangung des akademischen Grades "Doktor-Ingenieur" mit dem Titel "Novel View Prediction Error as a Quality Metric for Image-Based Modeling and Rendering" selbständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben. Ich habe bisher noch keinen Promotionsversuch unternommen.

Darmstadt, den 6. Juli 2017

Michael Wächter

Curriculum Vitae[73]

2011 – 2017	Research Assistant, Graphics, Capture and Massively Parallel Computing group, TU Darmstadt, Darmstadt
2015	Internship at Microsoft Research, Redmond, USA
2009 – 2010	Japanese and computer science, Keiō University, Tokyo, Japan
2008 – 2011	Master of Science, computer science, TU Darmstadt, Darmstadt
2005 – 2008	Bachelor of Science, computer science, TU Darmstadt, Darmstadt

[72] Gemäß § 9 Abs. 1 der Promotionsordnung der Technischen Universität Darmstadt
[73] Gemäß § 20 Abs. 3 der Promotionsordnung der Technischen Universität Darmstadt